

【特許請求の範囲】

【請求項1】 ファイルシステムを有するコンピュータシステムにおいて、ファイル内のデータを暗号化又は解読する方法であって、

ファイルが暗号化されたとして指定されることを示す情報を、ファイルシステムにおいて受信するステップと、

ファイルに関連付けられた暗号化キーを受信するステップと、

ファイルデータを不揮発性記憶装置に書き込むための要求を受信し、かつファイルデータを受信し、応答して、暗号化キーを使用して、ファイルデータを暗号化ファイルデータに暗号化し、暗号化ファイルデータを不揮発性記憶装置に書き込み、ファイルに関連して不揮発性記憶装置に暗号化キー情報を書き込むステップと、

不揮発性記憶装置からファイルデータを読み取るための要求を受信し、応答して、不揮発性記憶装置から暗号化ファイルデータを読み取り、暗号化キーを使用して、暗号化ファイルデータを解読ファイルデータに解読し、解読ファイルデータを戻すステップとを具えたことを特徴とする方法。

【請求項2】 ファイルに関連して不揮発性記憶装置に暗号化キー情報を書き込むステップの前に、暗号化キーが少なくとも1つのパブリックキーにより暗号化されることを特徴とする請求項1に記載の方法。

【請求項3】 プライベートキーにより暗号化キー情報を解読するステップをさらに具備することを特徴とする請求項2に記載の方法。

【請求項4】 暗号化キーが複数のパブリックキーにより複数の暗号化キーに暗号化され、暗号化キーの各々がファイルに記憶されることを特徴とする請求項2に記載の方法。

【請求項5】 少なくとも1つの暗号化キーがユーザのパブリックキーにより暗号化され、少なくとも1つの他の暗号化キーが、回復エージェントのパブリックキーにより暗号化されることを特徴とする請求項4に記載の方法。

【請求項6】 各ユーザのパブリックキーにより暗号化されたファイル暗号化キーが、各回復エージェントのパブリックキーにより暗号化されたファイル暗号化キーとは別個のフィールド内に記憶されることを特徴とする請求項5に記載

の方法。

【請求項7】 ファイルに記憶された暗号化キー情報の保全性を確かめるステップをさらに具えたことを特徴とする請求項6に記載の方法。

【請求項8】 ファイル暗号化キーを確かめるステップが、ユーザのパブリックキーによりファイル暗号化キーをハッシュするステップと、ユーザのパブリックキーによりハッシュされたファイル暗号化キーとファイル暗号化キーを暗号化するステップと、ハッシュされたファイル暗号化キーとファイル暗号化キーをファイルに記憶するステップとを含むことを特徴とする請求項7に記載の方法。

【請求項9】 ユーザ又は回復エージェントのいずれかのプライベートキーを受信するステップと、一致が検出されるまで、プライベートキーを使用することにより、ファイルに記憶された暗号化されたファイル暗号化キーを走査するステップとをさらに具えたことを特徴とする請求項5に記載の方法。

【請求項10】 回復ポリシーがファイルに関連して記憶され、回復ポリシーの保全性を確かめるステップをさらに具えたことを特徴とする請求項1に記載の方法。

【請求項11】 回復ポリシーが、回復エージェント情報を有するファイルのフィールド内に記憶され、回復ポリシーの保全性を確かめるステップが、前記フィールドの少なくとも一部分をハッシュするステップと、ハッシュされた一部分をファイル暗号化キーにより暗号化するステップと、暗号化されハッシュされた一部分を情報として前記フィールドに記憶するステップと、情報を検索するステップと、情報を現情報と比較するステップとを含むことを特徴とする請求項9に記載の方法。

【請求項12】 乱数を発生するステップと、ファイル暗号化キーをそれに基づかせるステップとをさらに具え、ファイル暗号化キーが少なくとも1つのパブリックキーにより暗号化され、ファイルに記憶されることを特徴とする請求項1に記載の方法。

【請求項13】 データを暗号化及び解読するステップがそれぞれ、関数のランタイムライブラリヘコールアウトを行うステップを含むことを特徴とする請求項1に記載の方法。

【請求項14】 ファイルシステムを有するコンピュータシステムにおいて、非暗号化ファイルデータ又は暗号化ファイルデータを読み取り、読み取られたデータを非暗号化ファイルデータとして戻す方法であって、不揮発性記憶装置からファイルデータを読み取るための要求を、ファイルシステムにおいて要求プログラムから受信するステップと、ファイルデータを読み取るステップと、ファイルデータが暗号化されているか否かを判定するステップと、ファイルデータが暗号化されていない場合、ファイルデータを要求プログラムに戻すステップと、ファイルデータが暗号化されている場合、パブリックキーにより暗号化されファイルに記憶されたファイル暗号化キーに基づいた、そのファイルに対する、ファイル暗号化キーを獲得するステップと、プライベートキーによりファイル暗号化キーを解読するステップと、ファイル暗号化キーとファイルデータを解読メカニズムに提供するステップと、ファイルデータを非暗号化ファイルデータに解読するステップと、非暗号化ファイルデータを要求プログラムに戻すステップとを具えたことを特徴とする方法。

【請求項15】 ファイルデータが暗号化されているかどうかを判定するステップが、ファイルデータに記憶された暗号化ビットをチェックするステップを含むことを特徴とする請求項14に記載の方法。

【請求項16】 不揮発性記憶装置からファイルデータを読み取る要求を受信するステップが、生データ読取りをファイルシステムに通知するステップを含み、データが不揮発性記憶装置から読み取られ、前記データを解読メカニズムに提供することなく戻されることを特徴とする請求項14に記載の方法。

【請求項17】 ファイル暗号化キーを確かめるステップをさらに具えたことを特徴とする請求項14に記載の方法。

【請求項18】 ファイル暗号化キーを確かめるステップが、ファイルに記憶された暗号化されたファイル暗号化キー情報を解読するステップを含むことを特徴とする請求項17に記載の方法。

【請求項19】 ファイル暗号化キーが、複数のパブリックキーにより、複数のファイル暗号化キーに別々に暗号化され、少なくとも1つのパブリックキーがユーザ用であり、別のキーが回復エージェント用であり、暗号化された各ファ

イル暗号化キーがファイルに記憶されることを特徴とする請求項14に記載の方法。

【請求項20】 ユーザ又は回復エージェントのいずれかのプライベートキーを受信するステップと、非暗号化ファイル暗号化キーに一致する解読されたファイル暗号化キーが検出されるまで、暗号化されたファイル暗号化キーをプライベートキーにより解読することにより、ファイルに記憶された暗号化されたファイル暗号化キーを走査するステップとをさらに具えたことを特徴とする請求項19に記載の方法。

【請求項21】 少なくとも1人の回復エージェントに関して記憶された情報の保全性を確かめるステップをさらに具えたことを特徴とする請求項19に記載の方法。

【請求項22】 ファイルシステムを有するコンピュータシステムにおいて、選択されたファイルデータを暗号化ファイルデータとして記憶する方法であって、ファイルデータと暗号化された時に書き込まれるファイルデータとを書き込む要求を、ファイルシステムにおいて受信するステップと、ファイル暗号化キーを使用することにより、ファイルデータを暗号化ファイルデータに暗号化するステップと、パブリックキーによりファイル暗号化キーを暗号化するステップと、暗号化されたファイルデータと暗号化されたファイル暗号化キーを不揮発性記憶装置に書き込むステップとを含むことを特徴とする方法。

【請求項23】 ファイルデータを不揮発性記憶装置に書き込む要求を受信するステップが、生データ書き込みをファイルシステムに通知するステップを含み、データは、前記データをさらに暗号化することなく不揮発性記憶装置に書き込まれることを特徴とする請求項22に記載の方法。

【請求項24】 ファイルシステムを有するコンピュータシステムにおいて、ファイルシステムによって不揮発性記憶装置に書き込まれたデータを暗号化するためのシステムであって、ファイルシステムに接続され、ファイル暗号化キーに基づいて非暗号化データを暗号化データに変換するための暗号化メカニズムであって、ファイルシステムがデータの少なくとも一部を暗号化データとして不揮発性記憶装置内のファイルに書き込む、ファイル暗号化キーを獲得するための手

段と、ファイルシステムが暗号化されたファイル暗号化キーをファイルに関連して不揮発性記憶装置に書き込む、ファイル暗号化キーを暗号化するための手段とを具えたことを特徴とするシステム。

【請求項25】 ファイルに関連して記憶されたファイル暗号化キーを検索するための手段をさらに具え、暗号化されたファイル暗号化キーを解読するための手段と、ファイルシステムに接続され、ファイル暗号化キーに基づいて暗号化データを解読データに変換するための解読メカニズムとを含むことを特徴とする請求項24に記載のシステム。

【請求項26】 ファイル暗号化キーを獲得するための手段が、乱数発生器を含むことを特徴とする請求項24に記載のシステム。

【請求項27】 ファイル暗号化キーを獲得するための手段が、暗号サービスへのインターフェースを含むことを特徴とする請求項24に記載のシステム。

【請求項28】 ファイル暗号化キーを暗号化するための手段が、少なくとも1人のユーザのパブリックキーにアクセスするための手段を含むことを特徴とする請求項25に記載のシステム。

【請求項29】 ファイル暗号化キーを解読するための手段が、少なくとも1人のユーザのパブリックキーに対応するプライベートキーを受信するための手段を含むことを特徴とする請求項28に記載のシステム。

【請求項30】 ファイル暗号化キーを暗号化するための手段が、少なくとも1人の回復エージェントのパブリックキーにアクセスするための手段を含むことを特徴とする請求項25に記載のシステム。

【請求項31】 ファイル暗号化キーを解読するための手段が、少なくとも1人の回復エージェントのパブリックキーに対応するプライベートキーを受信するための手段を含むことを特徴とする請求項28に記載のシステム。

【請求項32】 暗号化メカニズムが暗号化ドライバを含むことを特徴とする請求項24に記載のシステム。

【請求項33】 暗号化メカニズムが、ファイルシステムを介して暗号化ドライバによって呼び出される関数のライブラリをさらに含むことを特徴とする請求項32に記載のシステム。

【請求項34】 暗号化ドライバと関数のライブラリの間で通信するための一般データ変換メカニズムをさらに具えたことを特徴とする請求項33に記載のシステム。

【請求項35】 暗号化メカニズムと解読メカニズムが共通の暗号化ドライバを含む請求項25に記載のシステム。

【請求項36】 検索されたファイル暗号化キーの保全性を確認するための手段をさらに具えたことを特徴とする請求項25に記載のシステム。

【請求項37】 確認するための手段が、ファイル暗号化キーに基づいてファイル暗号化情報を記憶するための手段を含むことを特徴とする請求項36に記載のシステム。

【請求項38】 暗号化メカニズムが暗号化ドライバを含み、ファイル暗号化キーを獲得するための手段が暗号サービスへのインターフェースを含み、サービスと暗号化ドライバの間の通信を暗号化するための手段をさらに具えたことを特徴とする請求項25に記載のシステム。

【請求項39】 ファイル暗号化キーを獲得するための手段が、ファイルに関連してファイル暗号化キー情報を一時的に記憶するためのキーコンテキストバッファを含むことを特徴とする請求項24に記載のシステム。

【請求項40】 ファイル暗号化キーを検索するための手段が、ファイルに関連してファイル暗号化キー情報を一時的に記憶するためのキーコンテキストバッファを含むことを特徴とする請求項25に記載のシステム。

【請求項41】 検索されたファイル暗号化キーの保全性を確認するための手段をさらに具えたことを特徴とする請求項25に記載のシステム。

【請求項42】 確かめるための手段が、ファイル暗号化キーに基づいてファイル暗号化情報を記憶するための手段を含むことを特徴とする請求項41に記載のシステム。

【発明の詳細な説明】

【0001】

(発明の分野)

本発明は、一般に、コンピュータ記憶技術に関し、さらに詳細には、コンピュータシステムのための暗号化ファイルシステム及び方法に関する。

【0002】

(発明の背景)

極秘データの保護は、コンピュータのユーザにとって非常に重要な問題になっている。たとえば、個人用記録あるいは顧客のクレジットカード番号のようなデータは、コンピュータに記憶され、そのコンピュータ（あるいは単に記憶媒体）が盗難にあった場合、窃盗犯は極秘情報にアクセスする。これはとくに、ラップトップコンピュータでは厄介であり、それはしばしば知能的な窃盗犯により盗難にであう。さらに、無権限の個人がコンピュータにアクセスし（多分そのコンピュータが単にオンラインである）、そのファイルの部分又は全部から情報を複写する時のように、記憶媒体が物理的に取られない時でさえ情報は横領される。極秘情報にアクセスする権限があるユーザは、情報がコピーされたことさえも気がつかないことがある。

【0003】

情報を保護するために、一形式の機密保持手順は、データを暗号化するものであり、そのため、データが悪意のある者の手に渡ったとしても、それはキーなしに読み取ることはできない。多くのアプリケーションレベルのプログラムは、ある形式のそのような暗号化を備える。しかし、ないよりはよいが、そのような現存する暗号化体系は、無数の問題をかかえる。

【0004】

1つの重大な問題は、暗号化データがなお、多数の攻撃を許していることである。さらに詳細には、暗号化機能を備えるアプリケーションは、パスワード／パスフレーズ導出キーを使用するが、辞書攻撃等のために、定義に対して軟弱である。さらに、攻撃は、時間とともに、また、特にハードウェアの改良とともに進んでおり、かつてあるアプリケーションによって安全とみなされたものが、もは

や安全ではないことがある。

【 0 0 0 5 】

また、ユーザは、キーを損失しがちである。損失キーの問題は、多様なユーザにキーを普及させることによって除去されるが、これは、さらに、安全性に妥協するものである。さらに、各ファイルは、異なるパスワードを有してもよいが、これは、再呼び出しを困難にする。したがって、便宜上、多数のユーザ達は、1つのファイルを暗号化するために使用されたと同一のパスワードキーにより多数のファイルを暗号化する。これにより、1つのファイルに対するキーを別の人に漏らすことはしばしば、多数の他のファイルに対するキーを、その人に不注意に与えることになる。さらに、1つ以上のファイルへのユーザアクセスを除去又は付加するために、各ファイル（および各ファイルのすべてのコピー）は、解読され、新キーで再暗号化され、その後、再配布されなければならない。

【 0 0 0 6 】

さらに別の重大な問題は、暗号化プロセスが不便であり、使用中各ファイルの暗号化及び解読を必要とすることである。結果として、多数の一般ユーザは、多様なファイルを暗号化することに悩むよりもむしろ、機密保持の心配を無視する傾向がある。

【 0 0 0 7 】

別の重大な問題は、大部分のアプリケーションは、ユーザが文書の作業中、一時的ファイルを作成することである。これらの一時的ファイルは、原文が保存された後、常に除去されるわけではなく、極秘データを攻撃され易いままにすることになる。また、最新のオペレーティングシステムでは、アプリケーションレベルの暗号化が、ユーザモードにおいてランし、こうして、ユーザの暗号化キーを含むすべてのデータは、ページファイルにされ、キーを検出する作業をきわめて容易にする。最近では、大部分のファイル暗号化アプリケーションは、組み込み暗号化アルゴリズムを有し、このため、異なる又は多様なアルゴリズムをサポートする時に、拡張性又は融通性がない。ここでのアルゴリズムは、アルゴリズムが時間の経過とともに改良される時、または、例えば、暗号化されるデータの秘密の度合に基づいて、速さ対強度の取捨選択をする、特定データに対する暗号化

アルゴリズムから選択するために、暗号化アルゴリズムをユーザに更新させるようなものである。

【 0 0 0 8 】

(発 明 の 目 的 及 び 要 約)

したがって、本発明の一般的目的は、上記の問題を実質的に解消するデータ暗号化のための改良システム及び方法を提供することにある。

【 0 0 0 9 】

本発明の別の目的は、ファイルシステムへ統合される上記の種類のシステム及び方法を提供することにある。

【 0 0 1 0 】

本発明の関連した目的は、暗号化及び解読が合法的ユーザに透過的に作用するシステム及び方法を提供することにある。

【 0 0 1 1 】

これらの目的を達成するために、極秘データを2人以上の合法的ユーザの間で共有する能力を提供し、ユーザのためのアクセスの付加及び除去が簡単である上記の特徴を有するシステム及び方法を提供することが、本発明の関連した目的である。

【 0 0 1 2 】

本発明のさらに別の目的は、ユーザがキーを失う時などの、暗号化データ回復の処置を取る強力な暗号法を提供することにある。

【 0 0 1 3 】

本発明のさらに別の目的は、融通性及び拡張性のある上記の種類のシステム及び方法を提供することにある。

【 0 0 1 4 】

簡単に言うと、本発明は、ファイル内のデータを暗号化又は解読するためのシステム及び方法を提供する。ユーザが、ファイル又はその親ディレクトリの暗号化を指定する時は常に、暗号化ファイルシステムは、ファイルに関連付けられた暗号化キーを受け取る。その後、システムが非暗号化テキストファイルデータをディスクに暗号化の方法で書き込む要求を受け取る時、ファイルシステムは、フ

ファイルデータを受け取り、暗号化キーを使用して、ファイルデータを暗号化ファイルデータに暗号化し、暗号化ファイルデータをディスクに書き込む。逆に、システムがディスクから暗号化ファイルデータを読み取る要求を受け取る時、ファイルシステムは、暗号化ファイルデータを読み取り、暗号化キーを使用して、読み取りデータを解読ファイルデータに解読し、解読ファイルデータをユーザに戻す。

【0015】

暗号化キーは、少なくとも1人のユーザと少なくとも1人の回復エージェント(recovery agent)のパブリックキーによって暗号化された乱数である。これらのキーは、ファイルに記憶され、これにより、ファイルは、ユーザ又は回復エージェントのいずれかのプライベートキーにより常に解読されることができる。

【0016】

他の目的及び利点は、図面を参照して、次の詳細な説明から明らかになるであろう。

【0017】

(好ましい実施形態の詳細な説明)

一般アーキテクチャ

まず、図面の図1を参照するに、本発明が取り入れられた、一般的に20で指定されたコンピュータシステム20が示されている。例示のコンピュータシステム20は、サーバ、ワークステーション、またはそれらの組み合わせであり、公知の方法で、1つ以上の他のコンピュータベースの資源へ接続される。もちろん、明らかになるように、発明は、特定形式のコンピュータ又はネットワークアーキテクチャに限定されず、スタンドアロン型パーソナルコンピュータ又は同等物へ取り入れてもよい。

【0018】

図1に示されたように、コンピュータシステム20は、オペレーティングシステム26をロードしたメモリ24に接続したプロセッサ22を有する。好ましくは、オペレーティングシステム26は、マイクロソフト社のWindows NTオペレーティングシステムである。コンピュータ20は、Windows

N T ファイルシステム 2 8 (N T F S 2 8) などのファイルシステム 2 8 を、オペレーティングシステム 2 6 に付随して又は含んでいる。しかし、理解できるように、本発明は、特定のオペレーティングシステム及び／又はファイルシステムに限定されないが、明確性のために、本発明は、以後、W i n d o w s N T と N T F S 2 8 を参照して説明される。メモリ 2 4 における少なくとも 1 つのアプリケーションプログラム 3 0 は、アプリケーションプログラミングインターフェース (A P I) 3 2 を介して、オペレーティングシステム 2 6 及びファイルシステム 2 8 とインターフェースする。

【 0 0 1 9 】

コンピュータシステム 2 0 はまた、1 つ以上のネットワーク装置、キーボード及び／又はマウスなどの 1 つ以上の入力装置 3 6 、及び／又はモニタ及び／又はスピーカなどの 1 つ以上の出力装置 3 8 へ、コンピュータシステム 2 0 を接続するための入出力 (I / O) 回路 3 4 を含む。コンピュータシステム 2 0 はまた、ハードディスクドライブなどの不揮発性記憶装置 4 0 を含む。理解できるように、不揮発性記憶装置 4 0 は、メモリ 2 4 のランダムアクセスメモリに関連して動作し、スワッピング技術により大量の仮想メモリを供給する。

【 0 0 2 0 】

ファイルシステム 2 8 は、ファイルの管理をするために、通信するためのデバイスドライバ 4 2 を介して不揮発性記憶装置 4 0 と接続され、一般に、(1) ファイルを記憶、参照、共有及び確保し、(2) ファイルデータにアクセスし、および (3) ファイル保全性を維持するための方法を含む。それにも拘わらず、ファイルシステム 2 8 とその関連オペレーティングシステムとの間に明確な区別が常にあるわけではなく、オペレーティングシステム内に含まれたファイルシステム 2 8 については特にそうである。したがって、ファイルシステム 2 8 に帰せられるプロセス又はステップのいずれか又はすべては、オペレーティングシステム 2 6 によって代替的に実行され、またその逆も行われる。

【 0 0 2 1 】

不揮発性記憶装置 4 0 は、多数のファイル 4 4₁ - 4 4_n を格納し、N T F S がファイルシステム 2 8 として役立つ時、データは属性データストリームにおいて

編成される。各ファイルに関連付けられたNTFSファイル制御ブロック (FCB) は、それに属するデータストリームを識別する情報を維持する。Windows NTとNTFS 2.8は、文献、Inside Windows NT、Helen Custer 著、Microsoft Press (1993)、およびInside the Windows NT File System、Helen Custer 著、Microsoft Press (1994)、において記載されるが、これらの内容は参照してここに編入されている。図1において示され、以下に記載されるように、本発明により、ファイル44₁-44₂の少なくとも幾つかは、暗号化データを格納する。

【 0 0 2 2 】

EFSドライバ

本発明の1つの態様により、ファイルを暗号化及び解読するために、図2に最良に示されるように、暗号化ファイルシステム (EFS) ドライバ46と、EFSランタイムライブラリ (FSRTL 48) 48と、EFSサービス50とを備え、暗号化ファイルシステムが、提供されている。EFSドライバ46は、NTFS 2.8の上に置かれ、EFSサービス50と通信し、ファイル暗号化キー、解読フィールド (下記)、および他のキー管理サービスを要求する。NTFS 2.8を介して、EFSドライバ46は、この情報をFSRTL 48へ伝達し、下記のように、開く、読み取る、書き込むおよび付加するなどのいろいろなファイルシステム動作を行う。発明により、一旦ファイルが暗号化され、あるいは暗号化ディレクトリにおいて記憶されると、暗号化及び解読プロセスは、ユーザに対して透過的に動作する。

【 0 0 2 3 】

NTFS 2.8が初期化された時、EFSドライバ46は、NTFS 2.8によって動的にロードされる。NTFS 2.8は、特定ボリュームに対し、そのボリュームにおけるファイルを作成／開くためのI/O要求パケットを最初に受信する時、または、そのボリュームにおけるファイルに暗号化関連動作を試行するファイル制御動作によりI/O要求パケットを受信する時、EFSドライバ46をNTFS 2.8に接続させる (すなわち、層状ドライバチェーンへ挿入される)。EFS

Sドライブ46の初期化中、EFSは、ファイルシステムランタイムコールバックルーチンをNTFS28に登録する。下記のように、NTFS28は、これらのFSRTL48ルーチンを使用し、ファイル暗号化関連サービスを得るためにコールバックする。

【 0 0 2 4 】

EFSドライブ46は、機密保持サブシステムの一部としてランするユーザーモードEFSサービス50と通信するためのサポートを提供する。初期化中、EFSドライブ46は、GenerateSessionKeyインターフェースを使用してEFSサービス50と通信し、ドライブ46とEFSサービス50の間で確実に通信するために使用される対称セッションキーを確立する。両者の間で通信されたすべてのデータは、このセッションキーを使用して暗号化される。このセッションキーはまた、EFSサービス50からのI/O制御を解読するために、FSRTL48への呼び出しによって使用される。

【 0 0 2 5 】

暗号化ファイルのオープン毎に、EFSドライブ46は、データ解読及びデータ回復フィールド（図3～図5、下記）を含むファイルメタデータを渡すことによりEFSサービス50と通信し、ファイル暗号化キーおよびファイルメタデータへの更新を取り戻す。ファイルメタデータは、ユーザが新キーに変更するので、更新され、あるいは回復エージェントのキーが更新される。EFSドライブ46は、この情報をFSRTL48へ伝達する。

【 0 0 2 6 】

非暗号化テキストファイル／ディレクトリの暗号化又は新暗号化ファイルの作成中、EFSドライブ46は、EFSサービス50と通信し、新ファイル暗号化キー、および暗号化ファイルに対する暗号化メタデータを得る。EFSドライブ46はまた、この情報をFSRTL48へ伝達する。

【 0 0 2 7 】

EFS FSRTL

FSRTL48は、暗号化ファイル及びディレクトリにおける読み取り、書き込み及び開くなどのファイルシステム28の各種の動作、ならびにディスクへの

読み書きが行われた時、ファイルデータを暗号化、解読及び回復する動作を取り扱うために、NTFS呼び出しを実施するモジュールである。このために、本発明は、NTFS 28とFSRTL 48の間のインターフェースを含む、呼び出しメカニズムを備える。以下により詳細に記載されるように、このインターフェースは、データを暗号化する上記のものを含む、データを変換する適切なライブラリ（およびドライバ）に対して一般的であり、こうして、NTFS 28とFSRTL 48の間のインターフェースは、より正確には、データ変換インターフェース 52として参照される。たとえば、インデックシングドライバは、このインターフェースを使用し、ディスクへのすべての書き込みを監視し、これらの書き込みに基づいて、インデックスを発生する。しかし、理解できるように、専用暗号化インターフェースが、代替的に提供されている。

【 0 0 2 8 】

EFSドライバ 46とFSRTL 48の間の動作は、EFS属性データ（解読データ及び回復フィールド）を、ファイル属性として書き込むことと、EFSサービス 50において計算されたファイル暗号化キーを、オープンファイルのコンテキストにおいてセットされるようにFSRTL 48へ通信することとを含む。それから、このファイルコンテキストは、不揮発性記憶装置 24へ及びからのファイルデータの書き込み及び読み取りにおける透過的な暗号化及び解読のために使用される。

【 0 0 2 9 】

データ変換インターフェース 52は、事実上任意の方法でデータを変換するエンジン又はドライバへインターフェースすることができるが、ここでの目的のためには、インターフェース 52は、データ暗号化を達成するために、EFSドライバ 46をファイルシステム 28へインターフェースするとして記載される。それにも拘わらず、データ変換インターフェースは、データ暗号化に限定されず、事実上任意の形式のデータ変更を達成するために適切である。しかし、現在、この変換モデルは、少なくとも元の非暗号化テキスト程度しかデータスペースを取らない適当なデータ変換をサポートする。いずれにせよ、EFSドライバ 46は、これらのコールバックをファイルシステム 28に登録し、これにより、ファイ

ルシステム 28 は、適切な時点において登録 EFS コールバック機能を使用して、ユーザが要求するいろいろな暗号化及び解読タスクを実行する。

【 0 0 3 0 】

発明にとって必ずしも必要ではないが、便宜上、FSRTL 48 は、EFS ドライバ 46 と共通ファイルにおいて格納される。事実、EFS ドライバ 46 と FSRTL 48 は単一コンポーネントとして実行されるが、それらは直接には通信せず、代わりに、NTFS ファイル制御呼び出しメカニズムを使用する。すなわち、EFS ドライバ 46 は、FSRTL 48 を有効に呼び出す。NTFS 呼び出しメカニズムの使用は、NTFS 28 がすべてのファイル動作に参加することを保証し、2人のユーザがロックされたような場合に、各ユーザは他方のファイルの解放を待つというような状態は回避される。

【 0 0 3 1 】

データ変換インターフェース 52 は、多数の関数ポインター、またはコールバックを含む。ファイルシステム 28 が使用する最初のコールバック、File Create コールバックは、登録 EFS 関数に、ストリームが作成又はオープンされていることを通知する。EFS ドライバ 46 がこの点において取る作用（たとえば、ファイルが現存ファイルであるならばユーザがアクセスを有するかを判定すること、あるいは新ファイルに対してメタデータストリームを得ること）は、以下でさらに詳細に記載される。

【 0 0 3 2 】

アプリケーションがファイルを開く又は作成する時、I/O サブシステム 56 は、ファイルがあるファイルシステム、例えば、NTFS 28 ファイルであるかを判定し、NTFS 28 へ要求を伝達する。NTFS 28 は、EFS がファイルに関与するか、例えば、ファイルが暗号化ディレクトリにおいて作成されるか、または、ストリームが作成されもしくは暗号化ファイルへ接続されるか、を判定する。NTFS 28 が、ファイルが EFS に関与することを判定し、EFS ドライバ 46 がドライバとして登録されたと判断するならば、NTFS 28 は、登録 EFS 関数、即ち、File Create コールバックを呼び出す。要求が現存ファイル内のファイルオープン要求であるならば、FSRTL 48 は、ファイル

属性からファイルメタデータを読み取り、コンテキストブロック（例えば、下記のように、EFSドライバ46によって以前に割り当てられた図7のブロック98₁）に記入し、その情報をEFSドライバ46へ返送する。呼び出しがNTFS28から返る時、EFSドライバ46は、メタデータ情報を取得し、EFSサービス50と通信し、メタデータからファイル暗号化キー60を抽出する。それから、この情報は、開かれているファイルについてキーコンテキスト96をセットする下記の別のFSRTL48インターフェース、FileControlにより、NTFS28へEFSドライバ46によって戻される。このキーコンテキスト96は、その後、ファイルが閉じられるまで、EFSドライバ46への今後の呼び出しのために、NTFS28によって保持される。ファイルメタデータが更新されるならば、更新メタデータはまた、NTFSコールバックを介して、登録EFS関数によって属性へ再書き込みされる。

【 0 0 3 3 】

新ファイルが作成されるならば、FileCreateコールは結果として、新ファイル暗号化キー及びメタデータに対する要求を、コンテキストバッファ98₁に記入するFSRTL48となる。それから、FSRTL48は、コンテキストバッファ98₁をEFSドライバ46へ返送する。EFSドライバ46は、この情報を取得し、EFSサービス50と通信し、EFSサービス50から新ファイル暗号化キーと新ファイルメタデータを獲得する。ファイル制御コールバック（下記）を使用して、EFSドライバ46は、この情報をFSRTL48へ返し、これにより、NtOfsコールを使用して、FSRTL48は、作成されているファイルについてキーコンテキスト98をセットし、ファイルメタデータを書き込む。NtOfs APIは、NTFS28関数コールのセットであり、EFSドライバ46はファイルシステム28を呼び、暗号化メタデータを含むデータストリームを操作することができる。

【 0 0 3 4 】

別のコールバック、FileSystemControl__1は、ユーザがファイルの暗号化状態（EFS__SET__ENCRYPT）を、暗号化又は解読と記してセットしている時、EFSドライバ46要求に応答して、NTFS28に

よって呼び出される。応答して、NTFS 28は、暗号化ビットをセット又はクリアし、そしてEFSドライバ46は、必要なキー記憶を生成する。EFS__SET__ENCRYPTはまた、非暗号化テキストファイルの暗号化が開始される時、EFSサービス50において生じ、これにより、ファイル状態は、暗号化が完了するまで、他の動作はファイルにおいて許容されないように、修正される。

【 0 0 3 5 】

NTFS 28はまた、EFSドライバ46からのいろいろな暗号化ドライバ特定ファイル制御要求により、FileSystemControl__2インターフェースを呼び出す。NTFS 28は、FSRTL 48へコールを単に伝達する以外に、これらのコールバックに関してなにも作用を行わないことに注意せよ。新又は更新ファイルメタデータの書き込みが望まれる時、EFSフィルタードライバ46から来るEFS__SET__ATTRIBUTEと、EFSドライバ46またはファイルメタデータを問い合わせるためのユーザモードアプリケーション30から来るEFS__GET__ATTRIBUTEとを、ファイル制御要求は含む。情報は、ユーザパブリックキー及び回復エージェントパブリックキー（下記）のリストを含み、ファイル暗号化キーを暗号化するために使用される。別の要求、EFS__DECRYPT__BEGINは、暗号化ファイルの解読を開始する時、EFSサービス50から来る。応答して、ファイルの状態は、解読が完了されるまで、他の動作はファイルにおいて許容されないように修正される。EFS__DEL__ATTRIBUTEは、全暗号化ファイルの解読を終了し、ファイルメタデータ及び関連属性を削除することを望む時、EFSサービス50において生ずる要求である。EFS__ENCRYPT__DONE要求はまた、ファイル暗号化を良好に完了する時、EFSサービス50から来る。ファイル状態は、この時点からの動作を可能にするように修正される。EFS__OVERWRITE__ATTRIBUTEは、暗号化ファイルがそのバックアップフォーマットから復元される時、EFSサービス50から来る。EFSサービス50は、ファイルにおいて現メタデータを重ね書きするために必要なファイルメタデータを供給する。この要求はまた、ファイルが復元されている間、読み取り又は書き込みが進行しないように、そのファイルに関連付けられたキーコンテキスト96の削除と関

連している。

【 0 0 3 6 】

F i l e S y s t e m C o n t r o l _ 2 インターフェースはまた、下記の F S C T L _ E N C R Y P T I O N _ F S C T L _ I O に応答して、ファイルシステム 2 8 によって呼び出される。これは、ファイルの状態が E F S ドライバ 4 6 の待機状態に対応することを、N T F S 2 8 が認識する時などの、E F S ドライバ 4 6 が N T F S 2 8 に E F S ドライバ 4 6 (自体) を呼び出させるための手段を備える。

【 0 0 3 7 】

ファイルシステム 2 8 は、暗号化ファイルに対してディスクからデータを読み取った後、および、それをユーザに戻す前に、コールバック、A f t e r R e a d P r o c e s s を直接に使用する。A f t e r R e a d P r o c e s s 関数は、このコールバックに応答して、進行中のデータを解読する。読み取り動作は、図 2 0 に関して、以下により詳細に記載される。

【 0 0 3 8 】

逆に、B e f o r e W r i t e P r o c e s s は、暗号化ファイルに対してデータをディスクに書き込む前に、ファイルシステム 2 8 によって呼び出される。関数は、このコールバックの結果として、データを暗号化する。書き込み動作は、図 2 1 に関して、以下により詳細に記載される。

【 0 0 3 9 】

C l e a n U p コールバックは、N T F S 2 8 がストリームに関連付けられた資源を解放している時、ファイルシステム 2 8 によって呼び出される。この時、E F S ドライバ 4 6 は、キー及び同等物を格納するように、使用中のメモリ資源を解放する。N T F S 2 8 が、ストリームにおける最後のクローズを受信する時、N T F S 2 8 は、キーコンテキスト 9 6 を含む、このオープンファイルの情報を得るためにメモリにおいて格納されたすべてのものを解放するために、その標準動作を行う。さらに、ファイルシステム 2 8 は、コンテキストブロック 9 8 により E F S ドライバ 4 6 を呼び出し、このファイル、例えばコンテキストブロック 9 8 、に対して消費しているメモリを解放する機会を与える。

【 0 0 4 0 】

A t t a c h V o l u m e コールバックは、上記のように（暗号化に係わる最初のユーザの動作中）ファイルシステム 28 によって呼び出される。応答して、E F S ドライバ 46 は、I / O サブシステムに、そのボリュームを表わす置対象に接続したい旨を通知し、これにより、そのボリュームに対して論理的に N T F S 28 の上に置かれ、その結果、I / O サブシステムは、E F S ドライバ 46 へ情報を最初に伝達する。D i s m o u n t V o l u m e は、ユーザがドライブを取り出し又は除去することを望むか、あるいはシステムが運転停止されているために、ボリュームが取り外されている時、ファイルシステム 28 によって呼び出される。D i s m o u n t V o l u m e コールに応答して、暗号化ドライバは、A t t a c h V o l u m e コールバック中になりに当てられたメモリ資源を解放する。しかし、E F S ドライバ 46 は、通常、分離され、ボリューム取り外しを I / O サブシステムによって通知された時、資源を解放するが、D i s m o u n t V o l u m e コールバックが提供され、融通性をさらに高めていることに注意されたい。

【 0 0 4 1 】

E F S サービス

E F S サービス 50 は、W i n d o w s N T 機密保持サブシステムの一部である。図 2 において E F S サービス 50 / ドライバ通信 54 として表わされたように、E F S サービス 50 は、L o c a l S e c u r i t y A u t h o r i t y (L S A) とカーネルモード機密保持参照モニタの間の現局所手順呼び出し通信ポートを使用し、E F S ドライバ 46 と通信する。ユーザモードにおいて、E F S サービス 50 は、マイクロソフトの暗号法 A P I 、クリプト A P I 58 とインターフェースし、ファイル暗号化キーを提供し、解読フィールド情報を生成する。

【 0 0 4 2 】

E F S サービス 50 はまた、暗号化、解読、回復のためのプログラミングインターフェースである W i n 3 2 A P I 32 のためのサポートを提供し、暗号化ファイルをインポート及びエクスポートするためのサポートを提供する。暗号化

ファイルのインポート及びエクスポートにより、ユーザは、下記のように、バックアップ、復元のような動作、および一般ファイル転送目的のために、ファイルを不透過（暗号化）データに変換することができる。Win32 API 32は、非暗号化テキストファイルを暗号化し、暗号文ファイルの解読又は回復、および（最初に解読することのない）暗号化ファイルのインポート及びエクスポートのためのプログラミングインターフェースを提供する。これらのAPI 32は、標準システムDLL、advapi32.dllにおいてサポートされる。

【 0 0 4 3 】

EFS サービス 50 は、セッションキーを生成し、EFS ドライバ 46 および FSRTL 48 とそれを交換することを含む、多数のサービスを提供する。EFS ドライバ 46 の要求に基づいて、EFS サービス 50 は、(CryptoAPI を使用して) 強力なセッションキーを暗号法により生成し、それをドライバおよび FSRTL 48 へ通信する。EFS サービス 50 はまた、EFS に対して定義されたユーザ及び回復エージェントのパブリックキーを使用して、ファイルに格納されたフィールド（図 3 ～ 図 5 を参照して以下に記載される、データ解読フィールド又は DDF、およびデータ回復フィールド又は DRF）においてファイル暗号化キーを生成する。EFS ドライバ 46 が新ファイル暗号化キーを要求する時、EFS サービス 50 は、CryptoAPI を使用して、この情報を生成し、それを EFS ドライバ 46 に戻す。

【 0 0 4 4 】

EFS サービス 50 はまた、ファイル暗号化キーを抽出する。すなわち、EFS ドライバ 46 がこの動作を要求する時、EFS ドライバ 46 は、DDF 及び DRF キーフィールドを含む、ファイルメタデータを供給する。その情報に基づいて、EFS サービス 50 は、DDF および（必要ならば）DRF キーフィールドを順次に探索し、ユーザのキーの名称を抽出し、クリプトAPI プロバイダー 58 を介してそのプライベート部分にアクセスする。うまくいったならば、（以下により詳細に記載されるように）、それは、暗号化されたファイル暗号化キーを解読のためにそのプロバイダーに伝達する。サービスは、解読が（同様に以下に記載されるように）正しいことを確かめ、また、ファイルメタデータにおいて使

用されたキーが最新であることを確かめる。キーが最新でないならば、サービスは、メタデータ（DDF及び／又はDRF）を再生成し、抽出されたファイル暗号化キーとメタデータを、EFSドライバ46に戻す。

【 0 0 4 5 】

EFSドライバ46がNTFS28によってロードされる時、それは、最初に、その構造を初期化し、メモリが常に利用可能であることを保証するために、あるスペースを確保する。その後、EFSドライバ46は、NTFS28に関して登録される。最後に、ドライバと同期するために、EFSドライバ46は、新イベントを作成しようとする。イベントがうまく作成されると、これは、EFSサービス50が初期化されず、EFSドライバ46が最初にロードされたことを示す。うまくいったならば、EFSドライバ46は、スレッドを作成し、信号が送られるのをイベントにおいて待つ。後に、イベントが送られる、すなわち、EFSサービス50が通信するためのレディ状態になる時、EFSドライバ46は、セッションキーを得るために、EFSサービス50を呼び出す。セッションキーはいったんEFSサービス50からEFSドライバ46へ転送され、EFSサービス50とEFSドライバ46は同期化される。イベントがうまく作成されなかったならば、それは、通常、EFSサービス50がすでに初期化されているためであり、このイベントでは、EFSドライバ46は、セッションキーを得るために、単に呼び出しを行うことに注意されたい。

【 0 0 4 6 】

EFSサービス50が最初にロードされた状況において、EFSサービス50は、新イベントを作成しようとする。イベントがうまく作成されるならば、EFSドライバ46は、初期化されていない。EFSサービス50は、イベントを待つことなく、セッションキーを生成する。後に、EFSドライバ46がロードされる時、EFSサービス50は、セッションキーを提供するために、EFSドライバ46によって呼び出される。EFSサービス50がセッションキーを提供する時、EFSサービス50は、EFSサービス50によって以前に作成されたイベントを閉じ、そしてEFSサービス50とEFSドライバ46は同期される。イベントがうまく作成されなかったならば、それは、通常、EFSドライバ46

がすでに初期化されたためであり、このイベントでは、EFSサービス50は、代わりに、イベントを開き、EFSサービス50がレディ状態であることをEFSドライバ46に知らせるようにイベントを送ることに注意されたい。その後、EFSサービス50は、EFSドライバ46によってセッションキーを要求され、同期が完了する。

【 0 0 4 7 】

W I N 3 2 システム A P I

とくに図22～図23に関して以下により詳細に記載されるように、EFSサービス50はまた、多数の他のユーザモードインターフェースを提供する。これらのインターフェースは、W I N 3 2 システム A P I と協働し、ユーザは、現存する非暗号化テキストファイルを暗号化ファイルへ変換し、暗号化ファイルを非暗号化テキストファイルへ変換し、バックアップ及び復元メカニズムを提供する等の、種々の動作を行うことができる。例示として、W I N 3 2 インターフェース32は、EFSサービス50と協働し、EFSの機能性を出し、ファイル暗号化（図22）を行うためにEFSサービス50によって提供されるインターフェース中にコールするラッパーであるE n c r y p t F i l e を含む。別のインターフェース、D e c r y p t F i l e は、ファイルの解読／回復（図23）を行うために、EFSサービス50によって提供されるインターフェースに同様にコールするラッパーである。

【 0 0 4 8 】

バックアップ／復元メカニズムはまた、システムA P I 中にコールされ、ユーザとバックアップオペレータは、解読なしに暗号化ファイルをバックアップすることができる。このために、O p e n R a w F i l e インターフェースにより、ユーザは、読み取りアクセスなしに、かつ、透過的な読み取り及び書込みを行うためのファイル暗号化キーをセットすることなく、暗号化ファイルを開くことができる。これらの動作に対して、N T F S 2 8 は、アクセスレベルを認識するが、このファイルに対するキーを探し出すために暗号化ドライバ46を呼び出さず、また、読み取りを解読し書込みを暗号化することもない。このインターフェースを介して開かれたファイルにおいて許される動作は、ファイル制御のみである

。こうして、ReadRawFile インターフェースにより、ユーザは、バックアップされ後に復元できる連続的な不透過ストリームとして、暗号化メタデータを含むすべてのデータを、ファイルから読み取ることができる。WriteRawFile インターフェースにより、ユーザは、暗号化ファイルを再作成するために、暗号化メタデータを含むすべてのデータを、そのバックアップからファイルに書き込むことができる。最後に、CloseRawFile が、提供され、ユーザは、OpenRawFile によって開かれた生ファイルを閉じることができる。

【 0 0 4 9 】

FileControl インターフェースにより、バックアップ及び復元メカニズムを提供するWin32API は、生の暗号化データを読み書きすることができる。生データの読み書きは、NTFS 28 とディスク（記憶装置）の間で直接的であることに注意されたい。EFS サービス 50 と EFS ドライバ 46 は、共通セッションキーを共有するので、EFS は連繫され、すべてのファイル制御は、（下記のように）確かめられる必要がある。ファイルをバックアップするために、Win32API は、EFS ファイル制御を介して EFS メタデータを読み取り、FileSystemControl_2 へ変換され、EFS ストリームが戻される。その後、NTFS 28 ファイル制御は、実ファイルデータを読み取るために呼び出され、データは不透過ストリームへパッケージされ、書き出される。ファイルを（漸次的に）復元するために、逆プロセスが行われる。EFS ファイル制御は、第 1 ストリームを識別するために呼び出され、別の EFS ファイル制御は、生データを書き戻すために呼び出される。

【 0 0 5 0 】

データ暗号化

図 3 ～ 図 5 において概念的に表わされているように、本発明は、パブリックキーベース体系を使用して、データ暗号化及び解読を実施する。このために、ファイルデータは、ファイル暗号化キー（FEK）60（図 3）による高速対称アルゴリズムを使用して、暗号化される。FEK 60 は、選択アルゴリズムによって必要とされるか、あるいはそうでなければ、アルゴリズムが可変長キーをサポート

トするならば必要とされる、ある長さのランダムに発生されたキーである。図3において表わされるように、乱数発生器62は、FEK60を生成する。FEK60を使用してファイルデータを暗号化するために、ファイルの非暗号化テキスト64は、適切なアルゴリズム（例えば、DES）を使用して、ファイル暗号化メカニズム66によって暗号化され、暗号化ファイル70に暗号化テキスト68として書き込まれる。

【0051】

本発明の別の態様により、図3に示されたように、ランダムに発生されたFEK60は、少なくとも1人のユーザのパブリックキー72によりそれ自体、暗号化され、データ解読フィールド（DDF）74と呼ばれる特殊なEFS属性で暗号化ファイル70に格納される。適切な暗号化アルゴリズム（例えば、RSA）を使用して、データ解読フィールド発生器76は、キー暗号化を行う。パブリックキーベース体系により、ユーザのキーペアのプライベート部分は、解読中、ただ使用される。すなわち、データ解読フィールド74における暗号化FEK60は、キーペアのプライベート部分を使用して、解読される。ユーザキーペアのプライベート部分84（図4）は、スマートカード及び／又は他の確実な記憶装置におけるように、別個の記憶場所に安全に記憶される。暗号化はまた、パスワード導出キーなどの、対称アルゴリズムを使用して行われるが、実行可能であるとしても、パスワードベース体系が辞書攻撃等のために本質的に弱いために、EFSは、好ましくは、そのような暗号化をサポートしないことに注意されたい。

【0052】

本発明の1つの態様により、図3において表わされるように、FEK60はまた、1つ以上の回復パブリックキー78を使用して、暗号化される。回復キー暗号化パブリックキー78は、下記の回復ポリシーによって指定されるように、回復エージェントとして知られた信頼ある人に属する。ユーザのパブリックキーを使用するFEKの暗号化と同様に、各回復キーペアのパブリック部分は、データ回復フィールド発生器80を使用して（例えば、ユーザのパブリックキーでFEK60を暗号化するために使用されるものと同じのアルゴリズムである必要はないが、RSAなどの適切な暗号化アルゴリズムを使用して）、FEK60を暗号

化するために使用される。暗号化 F E K のこのリストは、データ回復フィールド (D R F) 8 2 と呼ばれる特殊な E F S 属性中にファイル 7 0 といっしょに同様に格納される。こうして、回復キーペアのパブリック部分のみが、D R F 8 2 において F E K 6 0 の暗号化のために必要とされる。ユーザは任意の時点においてファイルを暗号化することを望むことから、適正な動作を容易にするために、これらのパブリック回復キーは、正常なファイルシステム 2 8 の動作のために E F S システムにおいて常時存在する。回復自体は、ユーザが組織を離れ、キーを失う等の時のみ必要とされるまれな動作であると想定される。結果として、回復エージェントはまた、スマートカード、フロッピーディスク、及び／又は他の確実な記憶装置においてキーのプライベート部分 9 0 (図 5) を記憶することができる。

【 0 0 5 3 】

本発明により、暗号化ファイルシステムアーキテクチャは、特定の暗号化アルゴリズムに限定されず、むしろ十分にアルゴリズム機敏性があり、いろいろな暗号化フェーズに対していずれかの暗号化アルゴリズムを使用する。結果として、本発明の暗号化ファイルシステムは、アルゴリズムが技術的に進歩するような場合に、ますます改良される暗号化アルゴリズムの使用を可能にする。さらに、ユーザは、暗号化及び解読の速度に対して（すなわち、より安全なアルゴリズムは一般により低速である）機密保持の程度を可変にしたアルゴリズム（すなわち、ユーザがその情報がどのくらい極秘か考えることに基づいて）を選択することを、利用可能なアルゴリズムから選定することができる。このように、上記の記述において、D E S は、ファイルデータを暗号化するために使用される 1 つのアルゴリズムであるが、R S A は、F E K を暗号化するために使用される。

【 0 0 5 4 】

データ解読

図 4 は、ユーザ解読プロセスを概念的に示す。ユーザのプライベートキー 8 4 (または回復エージェントのプライベートキー 9 0) は、データ解読フィールド 7 4 において格納された対応する暗号化された F E K 項目を解読するために使用される。キーの解読を達成するために、D D F 7 4 (および必要ならば、D R F

82) における各暗号化 F E K 項目、およびユーザ又は回復エージェントのプライベートキーは、F E K 60 を適正に解読する一致が見いだされるまで、(E F S サービス 50 における) 抽出メカニズム 86 へ対話式に送られる。そこから、F E K 60 は、暗号化テキスト 68 を非暗号化テキスト 64 に解読するために公知の方法で適切に対応する解読アルゴリズムを使用するファイル解読メカニズム 88 へ送られる。多様な解読アルゴリズムが利用可能である時、データ解読ワールドは、ファイル解読メカニズム 88 が正しい解読アルゴリズムを使用するように、暗号化アルゴリズムについての情報を記憶することに注意されたい。いくつかの各ファイルは、それ自身のアルゴリズムを有するが、唯一の暗号化アルゴリズムは、ファイル毎に使用される。

【 0 0 5 5 】

ファイルが開いている間に、解読された F E K 60 は、ファイル 70 に関連してファイルシステム 28 によって保存される。図 6 に示されたように、N T F S 28 により、ストリーム制御ブロック 94 は、各オープンファイルに対するファイル情報を維持し、そして暗号化ファイルに対応する各ストリーム制御ブロック (例えば、94_i) は、キーコンテキスト (例えば、96_i) を指している。キーコンテキスト 96 は、それぞれ、ディスクへの書き込み及び読み取り中、ファイルを暗号化及び解読するために必要な情報を維持する。以下にさらに詳細に記載されるように、F E K 60 は、ブロック毎のベースでファイルデータ読み取りを解読するために使用される。すなわち、大きなファイルへのランダムアクセスは、そのファイルに対してディスクから読み取られた特定ブロックのみを解読する。全ファイルは、必ずしも解読されとは限らない。

【 0 0 5 6 】

ファイル回復

図 5 は、回復プロセスを概念的に示す。回復プロセスは、D R F 82 において F E K 60 を解読するために、プロセスが回復エージェントのプライベートキー 90 を使用することを除いて、ユーザ解読と同様である。結果的に、D D F 74 において一致は見いだされず、こうして、一致に対する探索は、D R F 82 へと継続する。回復を始めるために、回復エージェントは、プライベートキー 90 を

提出し、そしてデータ回復フィールド抽出メカニズム92（好ましくは、上記の図4のデータ解読フィールド抽出メカニズム86と同一）は、DDF74を探索するために、エージェントのプライベートキー90を対話式に使用する。その時、それは回復エージェントであり、ユーザではないために、DDF74において一致は見いだされず、こうして、抽出メカニズムは、DRF82を探索し続ける。

【 0 0 5 7 】

正常なユーザオープンであるか、又は回復のためのオープンであるかに拘わらず、現コンテキストにおいてキーがいったん見いだされると（クリプトAPI58は各ユーザに対してキーのセットを維持する）、キーは、そのキーで解読された既知情報とそれを比較することにより、確かめられる。さらに詳細には、暗号化の時点において、ユーザのパブリックキーは、ファイルのFEKに付加され、その後、既知情報としてFEK60により暗号化される。見いだされたキーが、記憶情報を解読し、既知情報に等しかったとするならば、キーが確かめられる。この体系は、強力な暗号化技術を提供するが、ファイルを回復する能力を、多数の可能な回復エージェントの1人に与え、これにより、回復手順を実施する際の冗長性と融通性を組織に提供する。

【 0 0 5 8 】

EFSは、こうして、「回復ポリシー」と呼ばれる、組み込みデータ回復サポートを備えている。好ましいシステムは、回復キーのコンフィギュレーションを実施し、システムが1つ以上の回復キーで構成される時のみ、使用可能であるように意図的に限定される。ファイル回復動作は、ユーザ又は他の回復エージェントのプライベートキーではなく、ランダムに発生されたファイル暗号化キー60のみを露呈させる。理解できるように、これは、従業員が組織を去った又はキーを失った後、従業員によって暗号化されたデータを、組織が回復する必要がある大部分のビジネス環境にとって理想的である。

【 0 0 5 9 】

また、EFSポリシーとしても公知な回復ポリシーは、Windows NTドメインのドメインコントローラにおいて規定され、これにより、ポリシーは、

そのドメインにおけるすべてのマシンにおいて施行される。ポリシーは、回復エージェントのパブリックキーを含む。結果として、回復ポリシーは、ドメイン管理者の制御下にだけあり、これにより、データの回復者を制御する。付加的機能として、家庭環境においてスタンドアロン型Windows NTワークステーションにおいて暗号化機能を使用できるようにするために、EFSは、回復キーを自動的に生成し、マシンキーとしてそれらを保存し、これにより、平均的なユーザに対する管理オーバーヘッドを縮小する。

【 0 0 6 0 】

ドメインにおいて、回復ポリシーは、ドメインにおけるマシンの各々へ送信され、これにより、接続されていない時さえも、ローカルマシンは、ローカルな機密保持権限LSAにおいてポリシーを維持する。このようにして、EFSは、接続されていない時、ファイルを暗号化するために動作する。マシンがドメインをつなぐ毎に、あるいは回復ポリシーが変化するならば、ポリシーは、接続時に、ドメイン内のマシンへ伝搬される。さらに、ファイルが開かれる毎に、そのファイルに対するメタデータは、回復ポリシーに対して比較され、回復ポリシーが変化したかを調べ、そうならば、メタデータは、新ユーザ及び／又は回復エージェントのパブリックキー情報により暗号化されたFEKで更新される。ハッシュ(MD5)を含む、保護は、回復ポリシーが悪意のユーザ等によって変更されないことを保証するために使用される。

【 0 0 6 1 】

たとえば、悪意のユーザは、DRF82を変更することにより、ファイルを回復不能にしたがっているかもしれない。これを検出するために、DRF82の部分を使用して、暗号法ハッシュ(MD5)が作成され、ファイルのFEK60を付され、DRF82に格納される。後に、ファイルが開かれ、FEK60が獲得される時、その部分は、FEK60で解読され、DRF82において格納された情報が一致するかを調べる。そうならば、格納された回復ポリシーは適正であり、そうでなければ、回復ポリシーは、ドメインの現回復ポリシーと置き換えられる。

【 0 0 6 2 】

マシンがドメインの一部でないならば、それはなお、ローカルの回復ポリシーを有するが、回復キーは、上記のようにマシン毎に発生され、かつ保持される。このようにして、EFSの下で暗号化されたすべてのファイルは、常に、ある回復ポリシーを有する。マシンが、後に、ドメインの一部になるならば、ローカルの回復ポリシーは、一掃され、ドメインポリシーと置き換えられる。

【 0 0 6 3 】

一般動作

発明の動作の説明に戻り、図10の流れ図から始めると、アプリケーション30が暗号化ファイルを作成する又は開くことを望む時、アプリケーション30は、まず、適切なAPI32を呼び出し、新ファイルが、暗号化ディレクトリにおいて作成され、または暗号化ファイルが開かれることを要求する。

【 0 0 6 4 】

図10に示すように、いったん作成又はオープン要求が受信されると、I/Oサブシステム56は、ステップ1000～1002において、適切なファイルシステム、例えばNTFS28、へIRPとして要求を伝達するように準備する。しかし、上記のように、IRPは、まず、図11のステップ1100においてEFSドライバ46によって受信され、IRPはオープン／作成動作を指定するものとして認識される。

【 0 0 6 5 】

EFSドライバ46は、図11に示されたように、前処理を行うことにより、EFSCreateFile動作を始める。図11のステップ1102によって表わされるように、EFSドライバ46は、このファイルに対してEFSコンテキストブロック98₁を割り当て、EFSコンテキストブロックチェーン98（図7）へそれを追加する。EFSドライバ46は、この時点において、ファイルがすでに暗号化されているかまたは暗号化されることになっているかどうかを判断できないために、ファイルは暗号化されていないとしても、EFSコンテキストブロック98₁は、各新ファイルに対して作成されることに注意されたい。EFSコンテキストブロック98₁は、「処理は必要でない」で初期化されるステータス情報を含み、IRPポインターは、現ファイルオブジェクトを指し、EFS

SメタデータストリームはNULLに初期化される。最後に、ステップ1104において、IRPはNTFS28に伝達される。

【 0 0 6 6 】

図10のステップ1006において示されたように、NTFS28がEFSドライバ46からIRPを受信し、それをNTFS28作成パケットとして認識する時、NTFS28は、作成／オープンIRPを取り扱う。図12は、NTFS28がIRPをいかに取り扱うかを一般的に示す。まず、ステップ1200によって表わされるように、IRPにおける情報は、現ストリームが開かれるか、またはファイル／ディレクトリにおける新ストリームが作成されるかを判定するために検査される。現ストリームが開かれるならば、NTFS28は、ステップ1202においてストリームを開く。ステップ1204において、NTFS28は、ファイル又はそのディレクトリが暗号化されるかを判定し、もしそうならば、ステップ1208において、図13を参照して以下に記載されたFSRTL48オープン／作成コールアウトを呼び出す。

【 0 0 6 7 】

逆に、新ストリームがステップ1200によって判定されたように作成されたならば、NTFS28は、次に、ステップ1206において、親ディレクトリが暗号化されるかを判定する。親が暗号化されるならば、ステップ1208において、NTFS28は、以下に記載されるように、FSRTL48オープン／作成コールアウトを呼び出す。親が暗号化される（ステップ1206）ことも、またファイル又はディレクトリが暗号化される（ステップ1204）ことも判定されないならば、NTFS28は、FSRTL48コールアウトを行わず、これにより、NTFS28は、EFSドライバ46へ戻る前に、ステップ1210までに必要とするタスクを単に行う。

【 0 0 6 8 】

EFS作成／オープンFSRTL48コールアウトのステップは、図13において一般に表現され、この場合、FSRTL48コールアウトは、ユーザによって要求されたアクセスの形式を最初に検査することから始める（ステップ1300）。ファイル又はディレクトリにおける現ストリームが、読み取り、書き込み

、付加又は実行 (R / W / A / E) アクセスなしに開かれるならば、呼び出しは、単純に良好とされ、暗号化／解読は必要とされない。たとえば、ユーザは、属性を読み取することを望み、そして属性は暗号化されない。そうでなければ、ステップ 1 3 0 2 において、F S R T L 4 8 は、(図 1 1 のステップ 1 1 0 2 において割り当てられた) このファイルに対応する適切なファイルオブジェクトに対して E F S コンテキストチェーン 9 8 を探索する。作成／オープンコールアウトは、以下に提示されたように、ファイル／ディレクトリの形式に基づいて、種々の動作を行う。

【 0 0 6 9 】

ファイルの形式が現ファイル (ステップ 1 3 0 4) であるならば、新ストリームが作成されたか、または現ストリームが開かれたために、F S R T L 4 8 が呼び出された。そうならば、ユーザは、確認される必要があり、そしてコールアウトプロセスは、図 1 4 のステップ 1 4 0 0 へ継続する。ステップ 1 4 0 0 において、ファイルからの E F S メタデータは、(N t O f s) A P I を使用して、読み取られる。それから、ステップ 1 4 0 2 において、読み取られたメタデータは、コンテキストブロック 9 8 においてセットされ、ブロックにおけるステータスは、「ユーザ確認が必要」を指示するために変更される。その後、キーコンテキスト 9 6 が、ステップ 1 4 0 4 においてチェックされ、そして N T F S キーコンテキスト 9 6 が N U L L であるならば、キーが必要とされる。キーコンテキスト 9 6 が N U L L であるならば (ステップ 1 4 0 4) 、このファイルは、読み取られず、こうして、解読ファイルデータがキャッシュメモリに存在する可能性はない。結果として、コンテキストブロックが、ステップ 1 4 0 6 において「キャッシュチェックは必要とされない」を指示するためにセットされる。最後に、新ストリームがステップ 1 4 0 8 によって判定されたように作成されたならば、コンテキストブロック 9 8 が、ステップ 1 4 1 0 において「暗号化ビットをオンにする」を指示するためにセットされる。

【 0 0 7 0 】

代わりに、ファイルの形式が新ファイル (図 1 3 のステップ 1 3 0 6) であるならば、新 F E K および E F S メタデータが必要とされる。最初に、ステップ 1

308において、EFSメタデータが、Ntfs APIを使用して、親ディレクトリから読み取られる。ステップ1310は、コンテキストブロック98,において読み取られたメタデータをセットし、そのブロックにおけるステータスを、「新ファイルEfsが必要とされる」、「暗号化ビットをオンにする（ステップ1312）」および「キャッシュチェックは必要とされない（ステップ1314）」に変更する。

【0071】

代わりに、ファイルオブジェクト形式が新ディレクトリ（ステップ1320）を指示するならば、新EFSメタデータのみが必要とされる。現在、ディレクトリにおけるストリームは暗号化されていないために、この場合においてFEKはない。したがって、親ディレクトリからのEFSメタデータは、（Ntfs APIを使用して）ステップ1322において読み取られる。それから、ステップ1324において、たったいま読み取られたメタデータが、コンテキストブロック98,においてセッドされ、そしてそのブロックにおけるステータスは、「新ディレクトリEfsが必要とされる」、「キャッシュチェックは必要とされない」（ステップ1326）、および「暗号化ビットをオンにする」（ステップ1328）に変更される。

【0072】

最後に、形式が現ディレクトリ（ステップ1332）を表わすならば、新ストリームが作成されたか、または現ストリームが開かれた。現在、ディレクトリデータストリームは暗号化されていないために、作用はなににも行われなない。しかし、ディレクトリストリームはまた、本発明の暗号化ファイルシステムを使用して、ファイルデータストリームが暗号化されるのと同様にして暗号化されることは、容易に理解される。

【0073】

図12において示されたように、ステップ1210において、コールアウトは、NTFS28へ復帰し、これにより、NTFS28は、それ自身の動作のいずれかを行う。ファイル／作成プロセスは、後処理のためにステップ1010（図10）のEFSフィルタードライバ46へ復帰する。

【 0 0 7 4 】

E F S 作成／オープンファイル後処理プロセスは、図 1 5 ～図 1 9 において表わされる。図 1 5 の最初のステップ 1 5 0 0 において、コンテキストブロック 9 8₁が、「キャッシュチェックは必要とされない」ステータスに対して評価される。キャッシュチェックが必要とされるならば、プロセスはステップ 1 5 0 2 に分岐し、ここで、呼び出し元の機密保持コンテキストが、E F S ストリーム中に記憶されたファイルに対する E F S I D とともに、E F S キャッシュによって使用され、このファイルが最近、ユーザによって良好に開かれたかチェックされる。そうならば、キャッシュは適切な情報をすでに含むために、呼び出しは、良好である。

【 0 0 7 5 】

キャッシュにおいて、そうでないならば、ステップ 1 5 0 4 において、読み取りデータ、書き込みデータ、付加データまたは実行アクセスが必要とされるかがチェックされる。これらのいずれも要求されず、新ストリームがステップ 1 5 0 6 において判定されたように作成され、コンテキストブロック 9 8₁が「暗号化ビットをオンにする」ステータス（ステップ 1 5 0 8）を指示するならば、E F S データストリームは必要とされず、解放される。ストリームにおける暗号化ビットのみが、オンにされる必要があり、ステップ 1 5 1 0 において行われる。後処理は、完了し、全プロセスは、図 1 0 のステップ 1 0 1 2 へ復帰する。

【 0 0 7 6 】

しかし、図 1 5 に関して識別された上記の状況のいずれも満足されないならば、異なる動作が、図 1 6 のステップ 1 6 0 0 から検査されるステータス情報に基づいて行われる必要がある。まず、コンテキストブロック 9 0₁におけるステータスが、ユーザ確認がステップ 1 6 0 0 において必要とされることを指示するならば、E F S ドライバ 4 6 は、ステップ 1 6 0 2 において（I R P において提供された）機密保持コンテキストの役をする、ステップ 1 6 0 4 において、E F S サービス 5 0 を呼び出し、F E K を要求するために E F S メタデータを伝達する。

【 0 0 7 7 】

ステップ1606において、EFSサービス50は、コンテキストの役をすることにより、呼び出しに応答し、EFSメタデータにおける情報を使用して、FEKを解読するために、ユーザのプライベートキーを調べる。EFSサービス50はまた、ユーザのキーが更新されたか、または、回復キーがステップ1608によって判定されたように更新されるならば、EFSメタデータを更新する（ステップ1610）。いずれにせよ、ステップ1612において、EFSサービス50は、FEK60の保全性を確認し、すべての情報をEFSドライバ46に戻す。さらに詳細には、保全性を確かめるために、キー保全性ブロックが、次のように構成される。

【0078】

[F (FEK , P u k) , FEK] P u k

【0079】

ここで、

F () は、適切なハッシュ関数である（好ましくはMD5）。

P u k は、ユーザのパブリックキーである。そして

[] P u k は、ユーザのパブリックキーによる暗号化を表記する。

【0080】

結果的に、妥当なFEKがユーザのパブリックキーで解読されたと考えられる時、上記のブロックが本情報により計算され、ファイルにおいて格納されたブロックと比較される。それらが一致するならば、キー保全性が確認される。

【0081】

代替的に、ユーザ確認が必要とされた（ステップ1600）ことをステータスが指示しないが、代わりに、新ファイルFEKが必要とされたことを指示したならば、ステップ1614は、図17のステップ1700へ分岐する。ステップ1700において、EFSドライバ46は、IRPにおける保護コンテキストの役をし、そしてステップ1702において、EFSサービス50を呼び出し、親ディレクトリのEFSメタデータを伝達し、新FEKおよびEFSメタデータを要求する。ステップ1704において、EFSサービス50は、コンテキストの役をし、ランダムFEK60を生成する。ユーザがステップ1706によって判定

されたように、キーを有さないならば、ステップ1708において、EFSサービス50は、ユーザのためのキーペアを自動生成する。最後に、ステップ1710は、ユーザのパブリックキーおよび回復エージェントのパブリックキーの下で暗号化されたFEK60により、EFSメタデータストリームを作成する。EFSサービス50はまた、親ディレクトリへのアクセスを許容されたユーザがまたファイルへのアクセスを有する（NTFS28のアクセス制御リストはそのようなアクセスを許容するとする）ように、親ディレクトリのEFSメタデータにおけるすべてのパブリックキーを使用して、FEK60を暗号化する。

【0082】

ステップ1600もまた1614も満足されなかったならば、後処理は、ステップ1616へ分岐し、EFS分岐は、新ディレクトリFEKが必要とされることを指示したかを判定する。そうならば、後処理は、図18のステップ1800へ分岐し、ここで、EFSドライバ46は、IRPにおける保護コンテキストの役をする。ステップ1802は、EFSサービス50を呼び出し、親ディレクトリのEFSメタデータを伝達し、新EFSメタデータを要求する。ディレクトリストリームはこの時点において暗号化されていないために、FEKは必要とされないことに注意されたい。しかし、将来には、ディレクトリストリームはまた、ファイルストリームが本発明により暗号化されると同様に暗号化される。

【0083】

いずれにせよ、ステップ1804において、EFSサービス50は、コンテキストの役をし、空きFEKを使用して、EFSメタデータストリームを作成する。ステップ1706は、ユーザがキーを有さないかをチェックし、有さないならば、ステップ1708において、EFSサービス50は、ユーザのためのキーペアを自動生成する。その後、ステップ1810において、空きFEKは、ユーザのパブリックキーと回復エージェントのパブリックキーの下で暗号化され、そしてFEKはまた、親ディレクトリへのアクセスを許容されたユーザが、アクセス制御リストがそのようなアクセスを許容するならば、ファイルへのアクセスを有するように、親ディレクトリのEFSメタデータにおけるすべてのパブリックキーを使用して、暗号化される。

【 0 0 8 4 】

最終的に、3つのステータスのどれがコンテキストにあるかに拘わらず、後処理は、適切なFSCTLを発行するために、図19のステップ1900において終局する。2つのFSCTL呼び出しが利用可能である。即ち、FSCTL__SET__ENCRYPTIONとFSCTL__ENCRYPTION__FSCTL__IOである。FSCTL__SET__ENCRYPTIONは、ストリームに対する暗号化ビットをオン又はオフにすることをNTFS28に告げる。FSCTL__ENCRYPTION__FSCTL__IOは、下記の多数の動作を行うために使用されるいろいろなFSCTLである。

【 0 0 8 5 】

このために、FSCTLが、図8と図9に示されたように、データ構造100を伴う。データ構造は、NTFS28が2つの形式のFSCTL呼び出しを区別することができるようなパブリックコード、並びに、EFSドライバ46及び／又はEFSサービス50に対する動作をさらに詳細に規定するためのEFSサブコードを含む。データ構造はまた、FEK情報（図8）またはファイルハンドル情報（図9）を指定するEFSデータを含み、時々、EFSメタデータを含む。機密保持の目的のために、EFSサブコードとEFSデータフィールドは、EFSサービス50が上記などの初期化された時確立されるセッションキーにより暗号化される。

【 0 0 8 6 】

使用において、FSCTL__SET__ENCRYPTIONが発行され、例えば、ファイルが暗号化ディレクトリに最初に入れられた時、そのファイルに対する暗号化ビットをオンにする。サブコードは、暗号化ビットがオン又はオフにされるかを指示する。そのような動作に対して、FEK60は、すでに既知であり、こうして、図8に示されたように、EFSデータは、FEKを含み、FEKは、セッションキーにより暗号化される。パブリックキーを除いてすべてが、セッションコードにより暗号化されることに注意されたい。データ構造100の保全性及びソースを検証するために、データ構造の暗号化部分が、解読される。その後、暗号化FEKが解読され、他のFEKと比較され、等しいならば、構造が確

認められる。EFSストリームは、利用できるならば、他の方法で知られたEFSメタデータと比較される。FEKは常に知られているわけではないために、同様の検証は、図9に示されたように、セッションキーとファイルハンドルを使用して、行われる。繰り返されるファイルハンドルは、8バイトの長さに等しいように、適切なフィールド（図9）において実際に使用される。

【 0 0 8 7 】

(FileSystemControl_1またはFileSystemControl_2を介し、NTFS28を介して伝達された)FSRTL48へのコールアウトは、付随するサブコード次第で、属性を重ね書きし、またはセットするために使用される。2ビットのサブコードは、重ね書き属性又はセット属性動作を表わす。EFSストリームが新ファイルに対して書き込まれる時、FILE__SYSTEM__CONTROL_1が、（上記のNTFS28によって行われるように）暗号化ビットをまたオンにするために、FSRTL48コールアウトで使用される。代替的に、FILE__SYSTEM__CONTROL_2は、暗号化ビットへの変更が必要とされない時、例えば、ユーザが単にユーザキーを変更しただけならば、コールアウトにより使用される。

【 0 0 8 8 】

いずれが使用されるかに拘わらず、1ビットのサブコードは、動作「EFSKeyBlobをセットする」を表わし、新暗号化キー情報が利用可能であり、適切なキーコンテキストへ入力される必要があることを、FSRTL48へ指示する。別のビットは、動作「EFSストリームを書き込む」を表現する。EFSストリームを書き込むは、ユーザ又は回復エージェントがパブリックキーを変更し、ファイルメタデータがデータ構造100におけるEFSメタデータで書き換えられる必要がある時等、EFSサービス50によって発行される。1つの他のサブコードは、「EFSストリームを得る」を表わし、ユーザがストリームをエクスポートするか、またはキー名称を知りたいことを望む時等、ファイルに対する現EFS属性が、EFSフィールドに書き込まれる。

【 0 0 8 9 】

こうして、図19へ戻ると、ステップ1900は、暗号化ビットをオンにする

ことが必要とされるかを検査し、そうならば、ステップ1902においてF S C T L _ S E T _ E N C Y R P T I O N制御を発行し、サブコードは、ビットをオンにすることを指示する。もちろん、セッションキー、ハンドル、同一物のハンドル及び暗号化コピーを含む、他のE F Sデータもまた、検証目的のためにデータ構造100にある。いずれにせよ、F S C T Lは、N T F S 28に達し、N T F S 28は、転回し、ストリームにおいて暗号化ビットを（すでにオンでないならばファイルにおいて）セットし、情報をF S R T L 48に伝達するためにF S R T Lコールアウトを呼び出す。

【 0 0 9 0 】

ステップ1900が満足されないならば、E N C R Y P T I O N _ F S C T L _ I O F S C T Lは、データ構造100における適切なサブコードで呼び出される必要がある。こうして、ステータスが「ユーザ確認が必要とされる」（ステップ1904）であるならば、サブコードが、ステップ1905においてK e y B l o bにセットされる。次に、メタデータがステップ1906によって判定されたように更新されたならば、ステップ1907は、ステップ1914における呼び出しの前に、書き込みE F Sストリームサブコードビットをセットする。そうでなければ、ステータスが「新ファイルF E Kが必要とされる」（ステップ1908）であるならば、サブコードは、ステップ1910においてK e y B l o bおよび書き込みE F Sストリームにセットされる。すなわち、両ビットがセットされる。これらのいずれでもないならば、ステータスは、「新ディレクトリF E Kが必要とされる」であり、そしてサブコードが、ステップ1912において書き込みE F Sストリームにセットされる。すなわち、他方のビットのみがセットされる。F S C T Lは、ステップ1914において発行される。

【 0 0 9 1 】

読み取りおよび書き込み

図20において最初に示されたように、本発明の読み書き動作の説明に戻ると、アプリケーション30が、オープンファイルからデータを読み取ることを要求する時（ステップ2000）、I/Oサブシステム56は、読み取り要求を受信し、それをI R Pとして適切なファイルシステム28、例えばN T F Sへ伝達す

る。しかし、まず、IRPは、EFSドライバ46によって受信され、IRPを読み取り要求に対応するとして認識し、結果として、ステップ2002においてIRPをNTFS28へ直接に送る。ステップ2004において、NTFS28は、他のファイルに対して非暗号化テキストデータを読み取るように、ディスクからバッファに暗号化データを読み取る。しかし、暗号化ファイルに対して、ファイルシステム28は、ステップ2006において、このファイルが暗号化されることを認識し、そしてステップ2008において、暗号化ドライバ46が以前に作成／オープンコールバックから返したキーコンテキスト96₁を記憶し、かつ得る。ステップ2010において、NTFS28は、AfterReadProcessコールバックを使用し、データを解読するために、暗号化コンテキストを含むデータおよび十分な情報を登録関数に提供する。一般に、情報は、ファイルへのオフセット、読み取りバッファへのポインター、読み取り長、およびキーを含む。ステップ2012において、暗号化ドライバ46は、データを解読し、それをファイルシステム28に返し、これにより、ステップ2014において、ファイルシステム28は、この非暗号化テキストをI/Oサブシステム56を介してアプリケーションへ通常の方法で戻す。しかし、ファイルインデクシングおよび他の情報を含むNTFS28の内部メタデータストリームは、暗号化されていないことに注意されたい。NTFS28は、ステップ2006においてこれらのストリームを認識し、これらの特定ストリームに対して、ステップ2008～2012を一時的にスキップする。

【0092】

図21において示されたように、アプリケーション30が、オープンファイルへデータを書き込むことを要求する時（ステップ2100）、I/Oサブシステム56は、書き込み要求を受信し、それをIRPとして適切なファイルシステム28、例えばNTFS、へ伝達する。しかし、最初に、IRPは、EFSドライバ46によって受信され、IRPを書き込み要求に対応するとして認識し、結果として、ステップ2102においてIRPをNTFS28へ直接に送る。ステップ2104において、NTFS28は、書き込まれるデータに変更がなされないように、書き込みデータを別個のバッファへ複写する。暗号化ファイルに対して

、ファイルシステム28は、ステップ2106において、このファイルが暗号化されることを認識し、ステップ2108において、暗号化ドライバ46が以前に作成／オープンコールバックから返したキーコンテキスト96₁を記憶し、かつ得る。ステップ2110において、NTFS28は、BeforeWriteProcessコールバックを使用し、データを暗号化するために、暗号化コンテキストを含むデータおよび十分な情報を、関数に与える。ステップ2112において、暗号化ドライバ46は、データを暗号化し、それをNTFS28へ返し、これにより、ステップ2114において、ファイルシステム28は、別個のバッファ内の今暗号化されたデータを、不揮発性記憶装置40へ通常の方法で、すなわち、任意の他のファイルに対する非暗号化テキストデータであるかのように、書き込む。再び、ファイルインデクシングと他の情報を含むNTFS28の内部メタデータストリームは、暗号化されていないことに注意されたい。NTFS28は、ステップ2106においてこれらのストリームを認識し、そしてこれらの特定ストリームに対してステップ2108～2112を一時的にスキップする。

【 0 0 9 3 】

ファイルAPI暗号化および解読

EFSはまた、格納されたファイルの暗号化及び解読を容易にするためにAPI32を備える。Win32 EncryptFile APIは、非暗号化テキストファイル／ディレクトリを暗号化するために使用される。図22において示されるように、このAPIにより、アプリケーション30（ユーザ）は、ステップ2200において暗号化するためのファイル名を提供し、この呼び出しは、動作を行うために、EFSサービス50に対する呼び出しへ変換される。ステップ2202において、EFSサービス50は、ユーザのためにファイルを開き、障害回復の目的のためにバックアップコピーを作成し、そしてステップ2204において、SET_ENCRYPTファイル制御（FSCTL）を発行することにより暗号化のためにそれをマークする。ステップ2206において、EFSサービス50は、コピー内の各ストリームからデータを読み取り、ステップ2208においてデータを原ファイルに書き戻す。書き込み動作中、暗号化ビットがセットされるので、データは、ディスクに書き込まれる前に、自動的に暗号化され

る。プロセスは、すべてのデータストリームが書き込まれるまで、ステップ2210を介して反復される。この呼び出しが良好に完了するならば（ステップ2212）、バックアップは、ステップ2214において削除され、そうでなければ、原ファイルは、ステップ2216において復元され、呼び出しは失敗になる。ディレクトリの場合に、暗号化するデータはないために、ディレクトリは単に暗号化されたと記される。上記のように、NTFS28の内部メタデータストリームは暗号化されていないことに注意されたい。

【0094】

WIN32 DecryptFile APIはまた、EFSサービス50によって提供され、暗号化ファイル／ディレクトリ動作の逆動作である。図23において示されたように、ステップ2300～2302において、EFSサービス50は、ファイル名を提供され、ユーザのためにファイルを開く。ステップ2306～2310は、すべてのストリームからデータを読み取り、解読は透過的に行われるために、これらのストリームを非暗号化テキストであるコピーに書き込む。ステップ2312において、サービスは、解読ファイル制御を発行し、メタデータを削除し、暗号化属性を除去する。その後、ステップ2314～2318によって示されるように、APIは、非暗号化テキストにおいて書き込まれた原文のコピーからのすべてのデータストリームを書き戻す。これが良好に完了するならば、コピーは、ステップ2322において削除され、そうでなければ、原文が、ステップ2324において復元される。ディレクトリの場合に、ディレクトリは、解読するデータはないために、メタデータと属性を削除するために、解読されたとして単に記される。

【0095】

理解されるように、EFSファイル暗号化は、ファイル毎又は全ディレクトリベースでサポートされる（しかしNTFS28はストリーム毎に動作する）。ディレクトリ暗号化は、透過的に実施される。すなわち、暗号化のためにマークされたディレクトリにおいて作成されたすべてのファイル（またはサブディレクトリ）は、自動的に暗号化して作成される。さらに、ファイル暗号化キーは、ファイル毎であり、ファイルシステムボリュームにおける移動／複写動作に対して安

全にする。現アプリケーションレベル体系とは異なり、以下に明らかになるように、暗号化及び解読動作が透過的に、かつバイトがディスクとの間で移動する最中に行われるために、ファイルは使用前に解読される必要はない。EFSは、暗号化ファイルを自動的に検出し、キー記憶装置からユーザのキーを見つけ出す。キー記憶装置のメカニズムは、CryptoAPIから挺入れされ、結果として、ユーザは、スマートカード及び／又はフロッピーディスクなどの、安全な装置においてキーを記憶する柔軟性を有する。

【 0 0 9 6 】

さらに、本発明により、EFSは、下側のファイルシステム28（例えば、NTFS）と協同し、これにより、EFSは、暗号化された一時的ファイルの（適正に開発されたアプリケーションプログラムによる）書込みを容易にする。そのようなアプリケーションプログラムにより、一時的ファイルが作成される時、原ファイルからの属性が一時的ファイルへ複写され、一時的コピーもまた暗号化される。さらに、EFSドライバ46は、Windows NTカーネルモードドライバであり、ファイル暗号化キーを記憶するために非ページプールを使用し、これにより、キーはそれをページファイルにしないことを保証する。

【 0 0 9 7 】

EFSアーキテクチャは、任意数の人員の間のファイル共有を、これらの人員のパブリックキーの単純な使用により可能にする。各ユーザは、プライベートキーを使用して、ファイルを独立に解読する。ユーザは、（構成されたパブリックキーペアを有するならば）容易に追加され、あるいは共有者の仲間から除去される。

【 0 0 9 8 】

スタンドアロン構成において、EFSにより、ユーザは、キーをセットする管理上の努力を払わずに、ファイルの暗号化／解読を開始することができる。すなわち、EFSは、構成されていないならば、ユーザに対するキーの自動生成をサポートする。ドメイン構成では、管理者は、EFSが動作可能になるために、1度ドメインポリシーをセットする必要がある。最後に、EFSはまた、ファイルサーバにおいて記憶された遠隔ファイルにおいて暗号化／解読をサポートする。

しかし、ディスクからいったん読み取られたデータは、進行中に解読され、このため、ファイル共有プロトコルが通信の暗号化を実施しないならば、回線を非暗号化テキストで移動する。すなわち、EFSは、通信暗号化ではなく、記憶暗号化を扱う。通信プロトコルは、そのような暗号化を設けるために使用される。

【 0 0 9 9 】

前述の詳細な説明から判明するように、データを暗号化するためのシステム及び方法が提供され、システム及び方法は、暗号化及び解読が合法的ユーザに透過的に作用するように、ファイルシステムへ統合される。システム及び方法は、2人以上の合法的ユーザの間で極秘データを共有する能力を提供し、ユーザに対する追加及び除去アクセスは、単純である。ユーザがキーを失う時などの、暗号化されたデータ回復を扱う強力な暗号法が、提供される。そのシステム及び方法は、柔軟性及び拡張性がある。

【 0 1 0 0 】

本発明は種々の変更及び代替的構成を可能とするが、ある例示の実施形態が、図面に示され、詳細に上記に記載された。しかし、開示された特定の形態に本発明を限定する意図はなく、逆に、本発明の趣旨及び範囲内に包含されるすべての変更、代替的構成、及び等価物を含むことを意図する。

【図面の簡単な説明】

【図 1】

本発明が取り入れられたコンピュータシステムを表わすブロック図である。

【図 2】

本発明のコンポーネントの一般アーキテクチャを表わすブロック図である。

【図 3】

データの暗号化において使用されたいろいろな論理コンポーネントを概念的に表わすブロック図である。

【図 4】

データの解読において使用されたいろいろな論理コンポーネントを概念的に表わすブロック図である。

【図 5】

暗号化データの回復において使用されたいろいろな論理コンポーネントを概念的に表わすブロック図である。

【図 6】

ファイル暗号化及び解読のためのキーコンテキスト情報を少なくとも部分的に含むファイルに関連したストリーム制御ブロックの表現図である。

【図 7】

暗号化ファイル及びディレクトリのための暗号化コンポーネントの間で情報を通信するために使用されるコンテキストチェーンの表現図である。

【図 8】

暗号化キー情報を含むファイル情報を相互に通信するための一定のコンポーネントによって使用されるデータ構造の表現図である。

【図 9】

暗号化キー情報を含むファイル情報を相互に通信するための一定のコンポーネントによって使用されるデータ構造の表現図である。

【図 10】

本発明の 1 つの態様により、ファイルを開く又は作成するための全制御フローを示す流れ図である。

【図 11】

ファイルを開く又は作成する一部として一般的に採用されるドライバ前処理ステップを表現する流れ図である。

【図 12】

ファイルオープン要求を取り扱うためにファイルシステムによって採用されるステップを表わす流れ図である。

【図 13】

本発明の 1 つの態様により、暗号化ファイルを開く又は作成するためにコールアウトによって採用される一般的なステップを表わす流れ図である。

【図 14】

本発明の 1 つの態様により、暗号化ファイルを開く又は作成するためにコールアウトによって採用される一般的なステップを表わす流れ図である。

【図 1 5】

ファイルを開く又は作成する一部として一般に採用されるドライバ後処理ステップを表わす流れ図である。

【図 1 6】

ファイルを開く又は作成する一部として一般に採用されるドライバ後処理ステップを表わす流れ図である。

【図 1 7】

ファイルを開く又は作成する一部として一般に採用されるドライバ後処理ステップを表わす流れ図である。

【図 1 8】

ファイルを開く又は作成する一部として一般に採用されるドライバ後処理ステップを表わす流れ図である。

【図 1 9】

ファイルを開く又は作成する一部として一般に採用されるドライバ後処理ステップを表わす流れ図である。

【図 2 0】

ファイル読み取り要求を取り扱うためにいろいろなコンポーネントによって採用されるステップを表わす流れ図である。

【図 2 1】

ファイル書き込み要求を取り扱うためにいろいろなコンポーネントによって採用されるステップを表わす流れ図である。

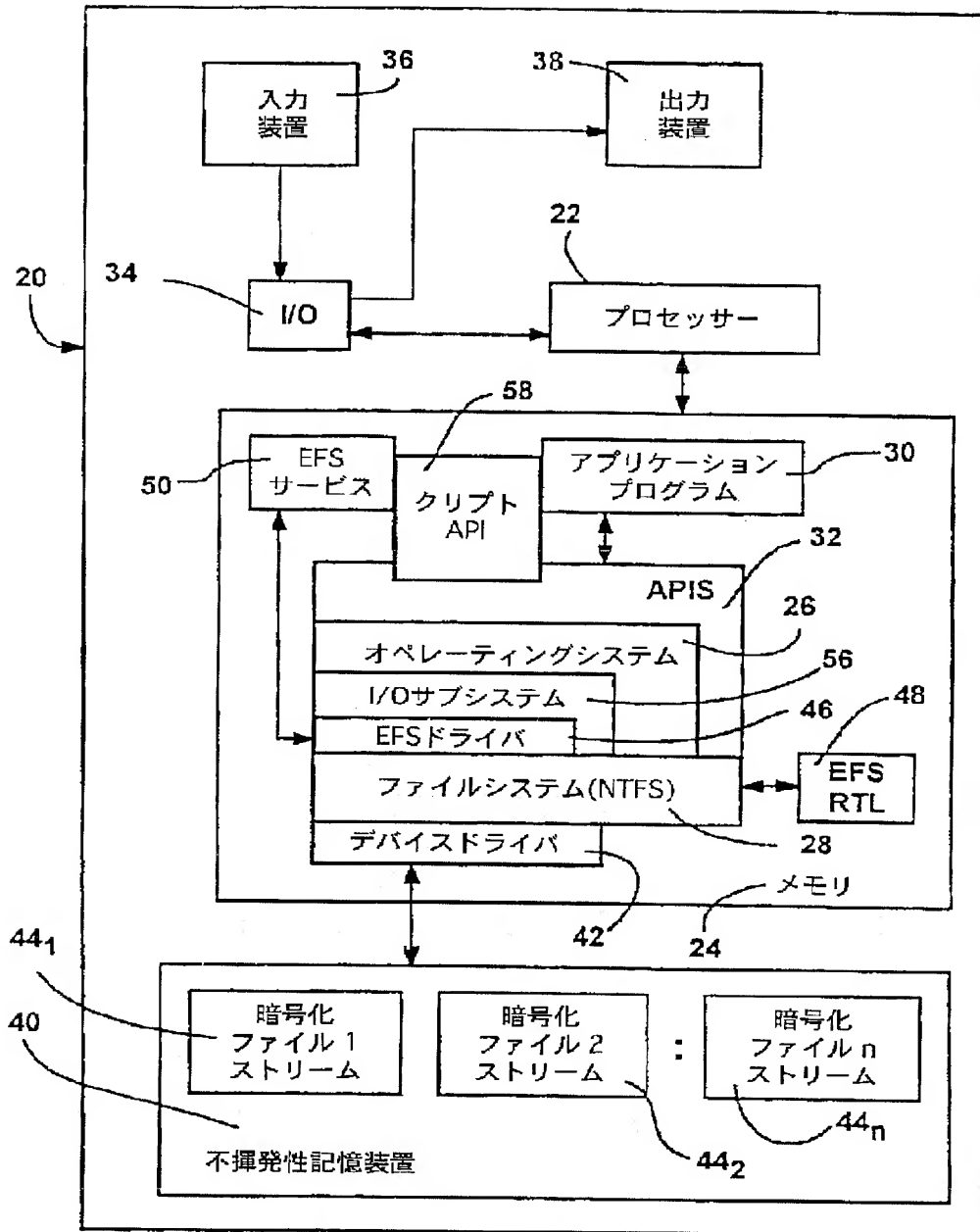
【図 2 2】

記憶非暗号化テキストファイルを暗号化するためのユーザ要求を取り扱うためにいろいろなコンポーネントによって採用されるステップを表わす流れ図である。

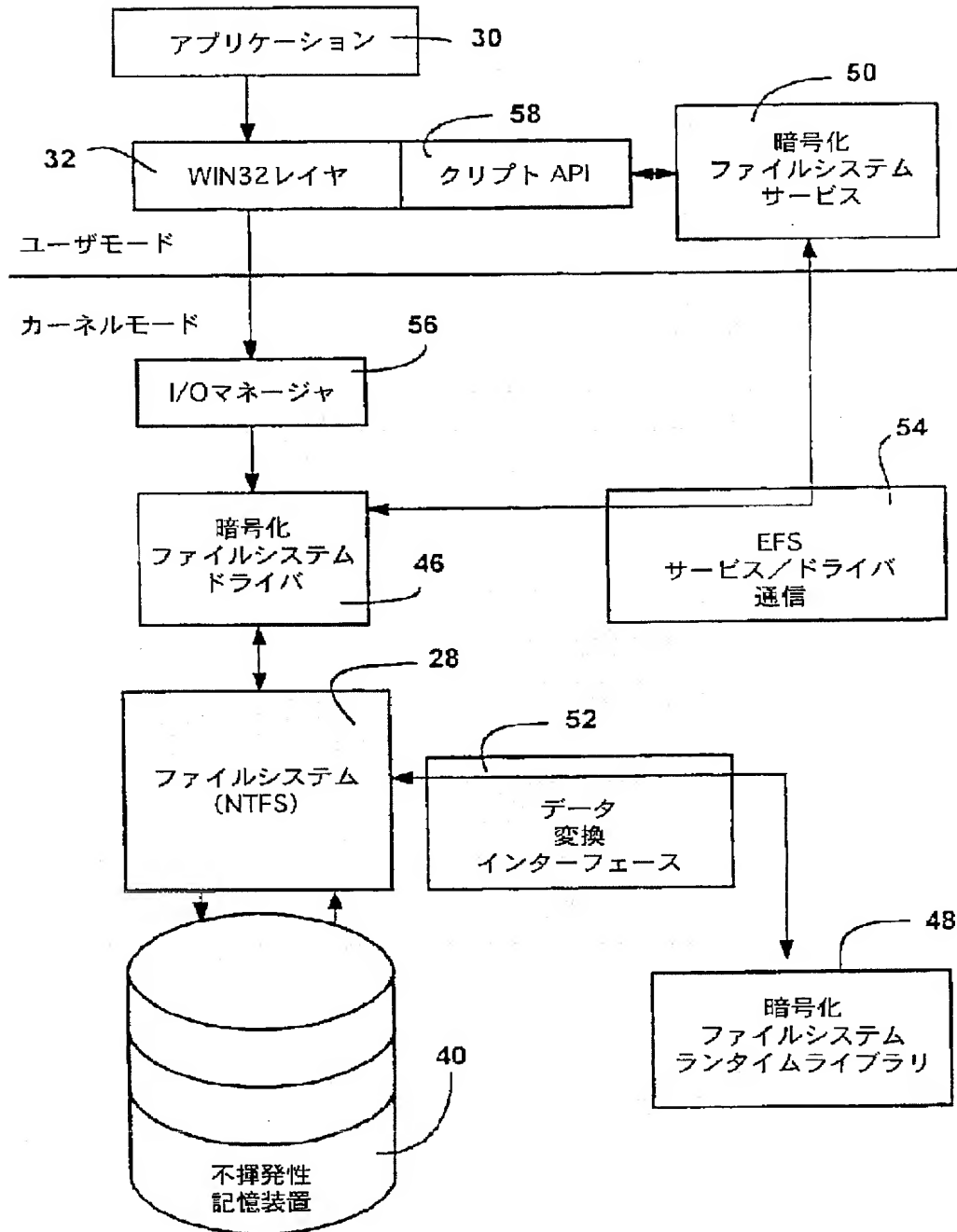
【図 2 3】

記憶暗号化ファイルを解読するためのユーザ要求を取り扱うためにいろいろなコンポーネントによって採用されるステップを表わす流れ図である。

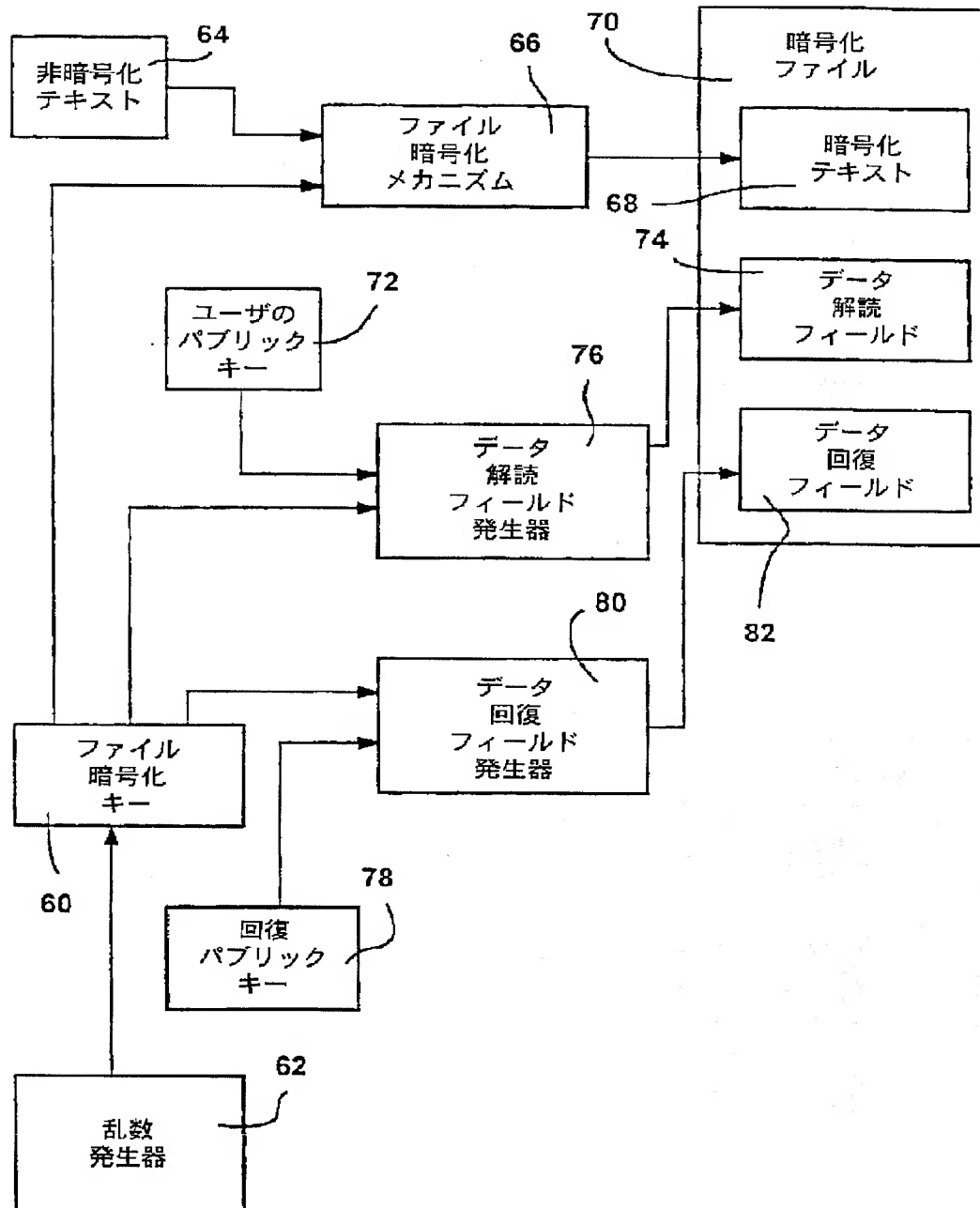
【 図 1 】



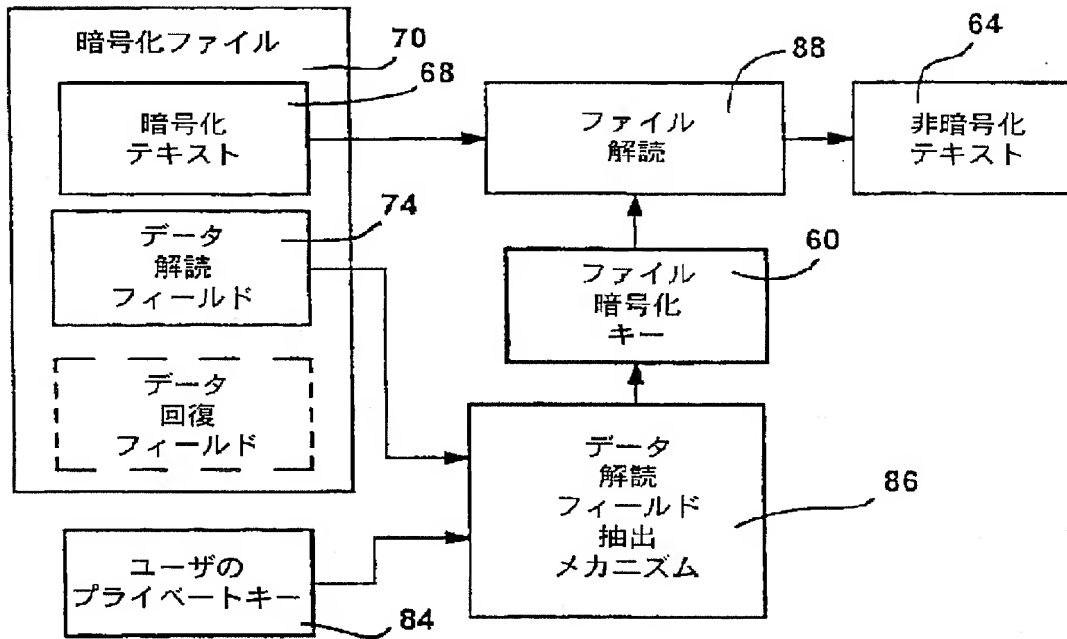
【図2】



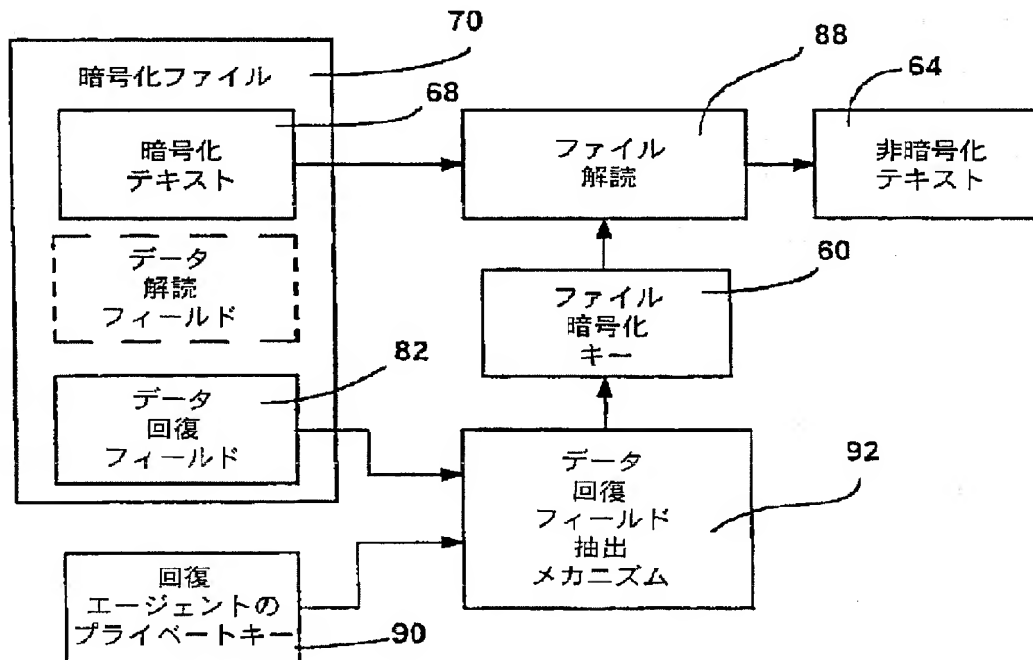
【 図 3 】



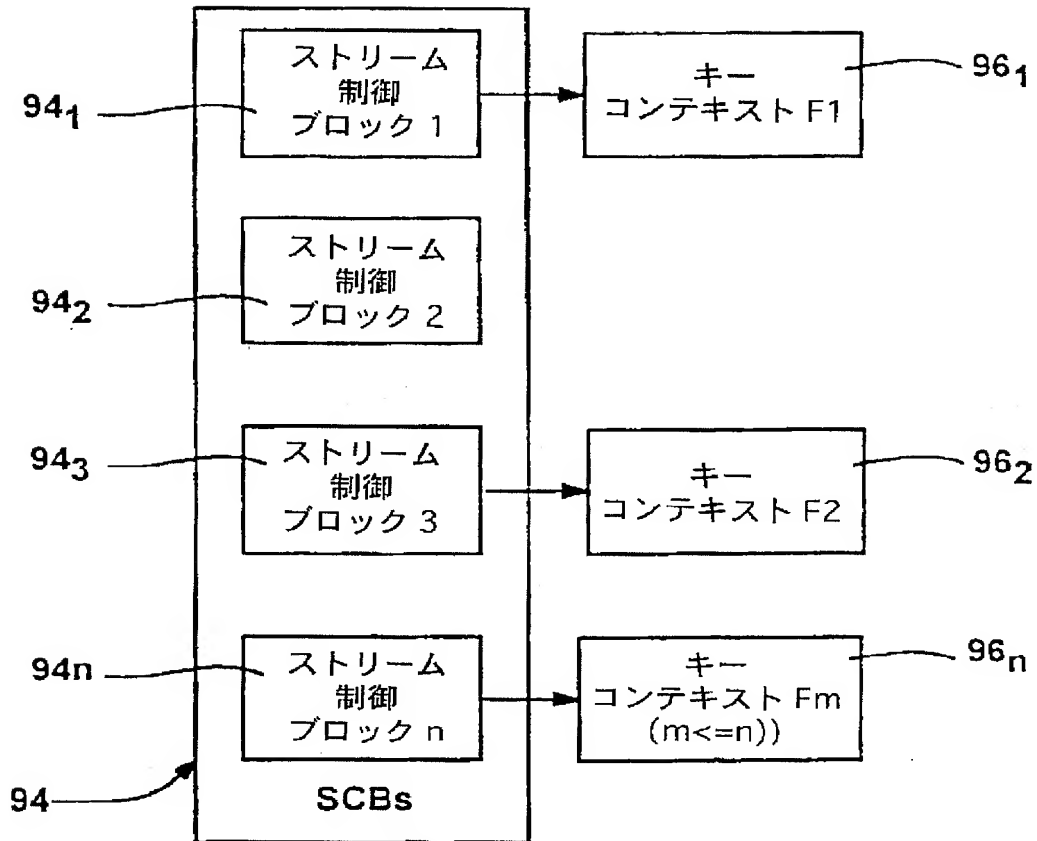
【 図 4 】



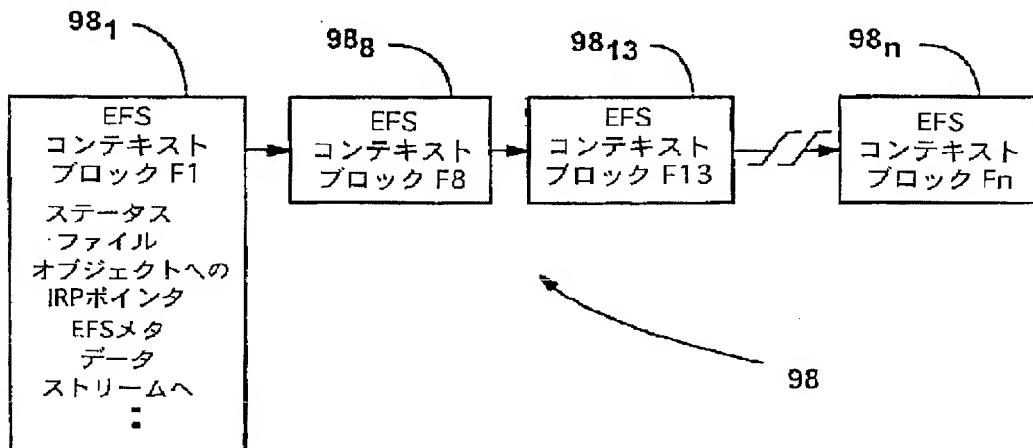
【 図 5 】



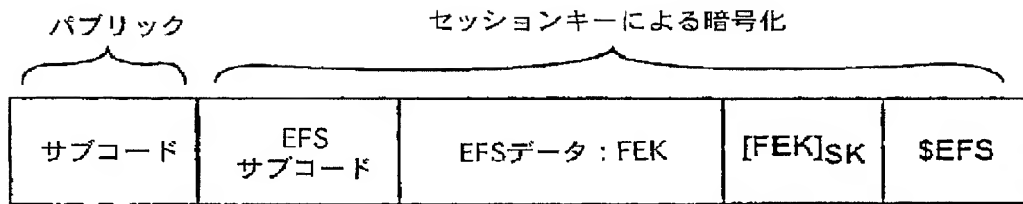
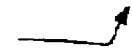
【 図 6 】



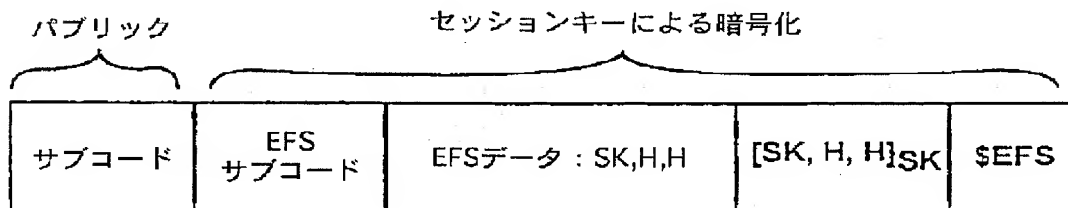
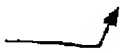
【 図 7 】



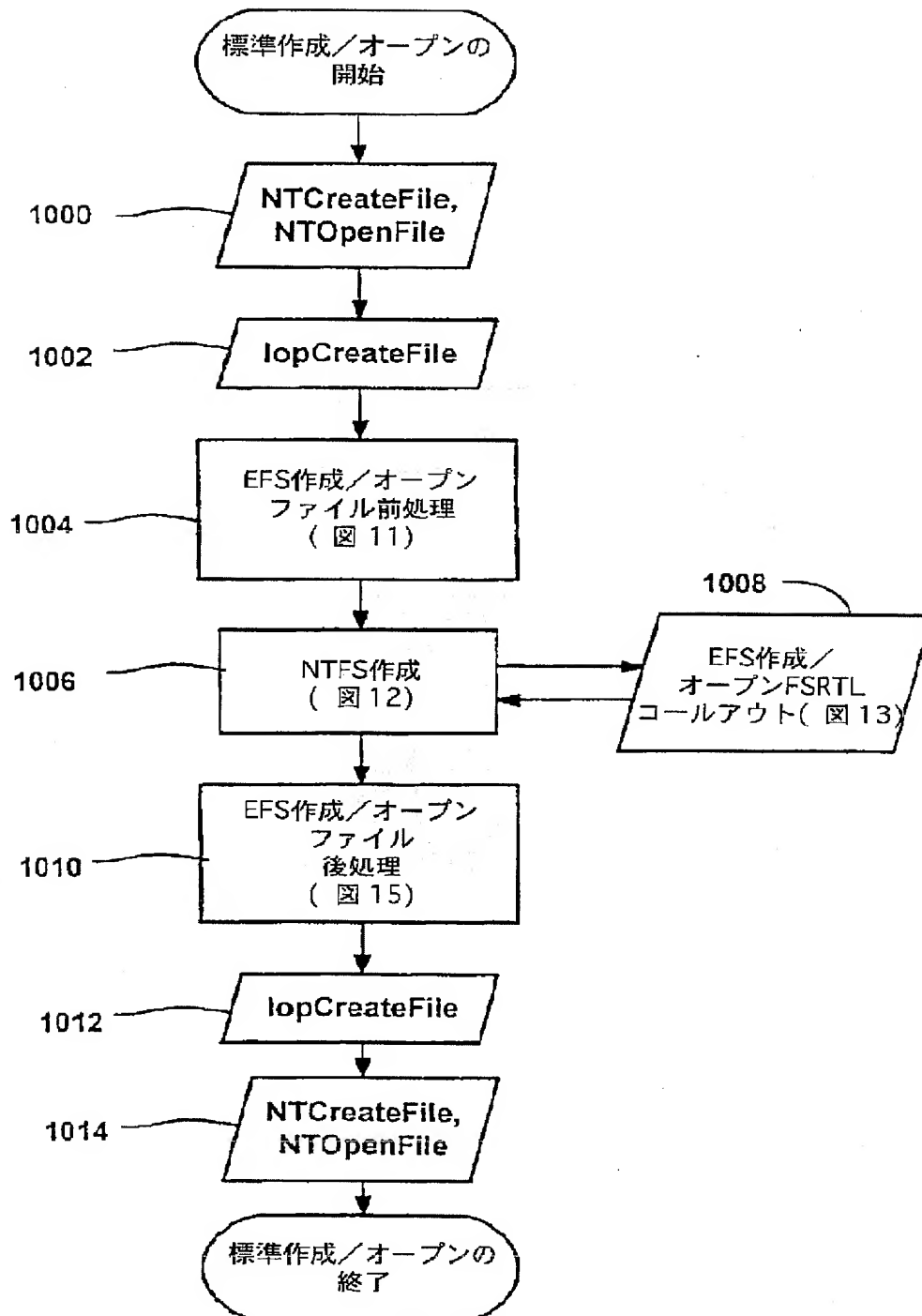
【 図 8 】

100 

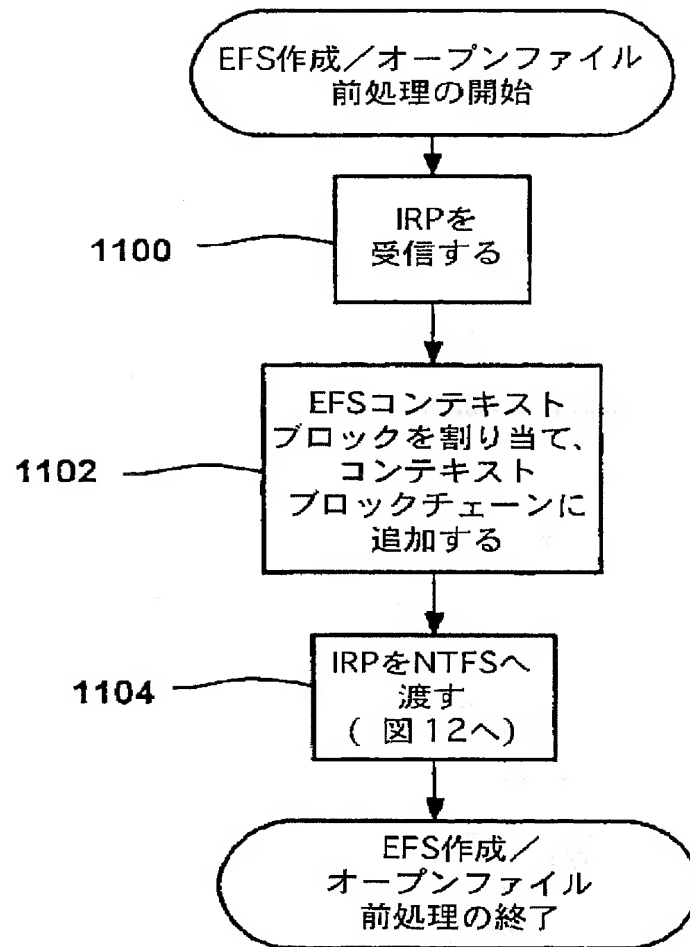
【 図 9 】

100 

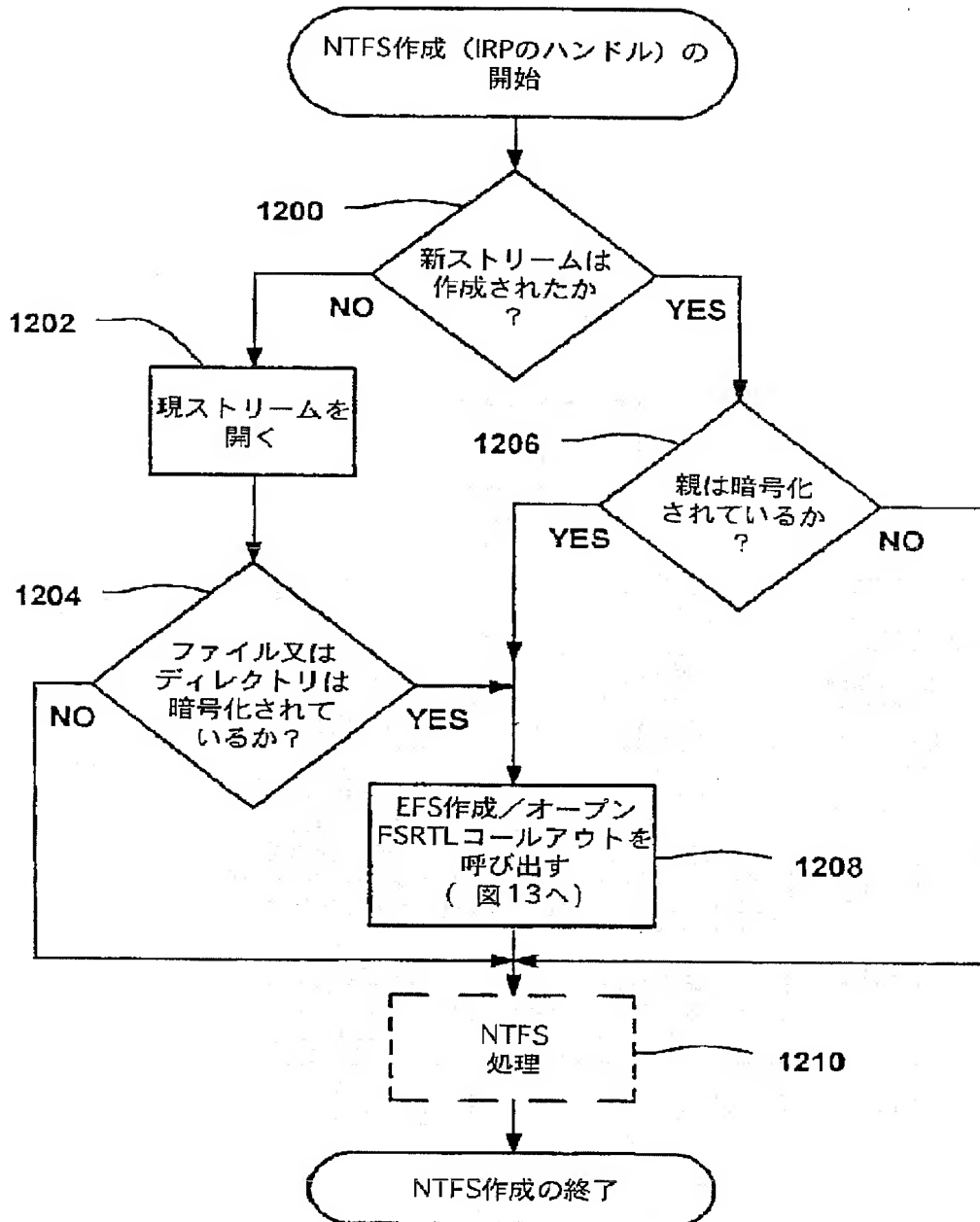
【図10】



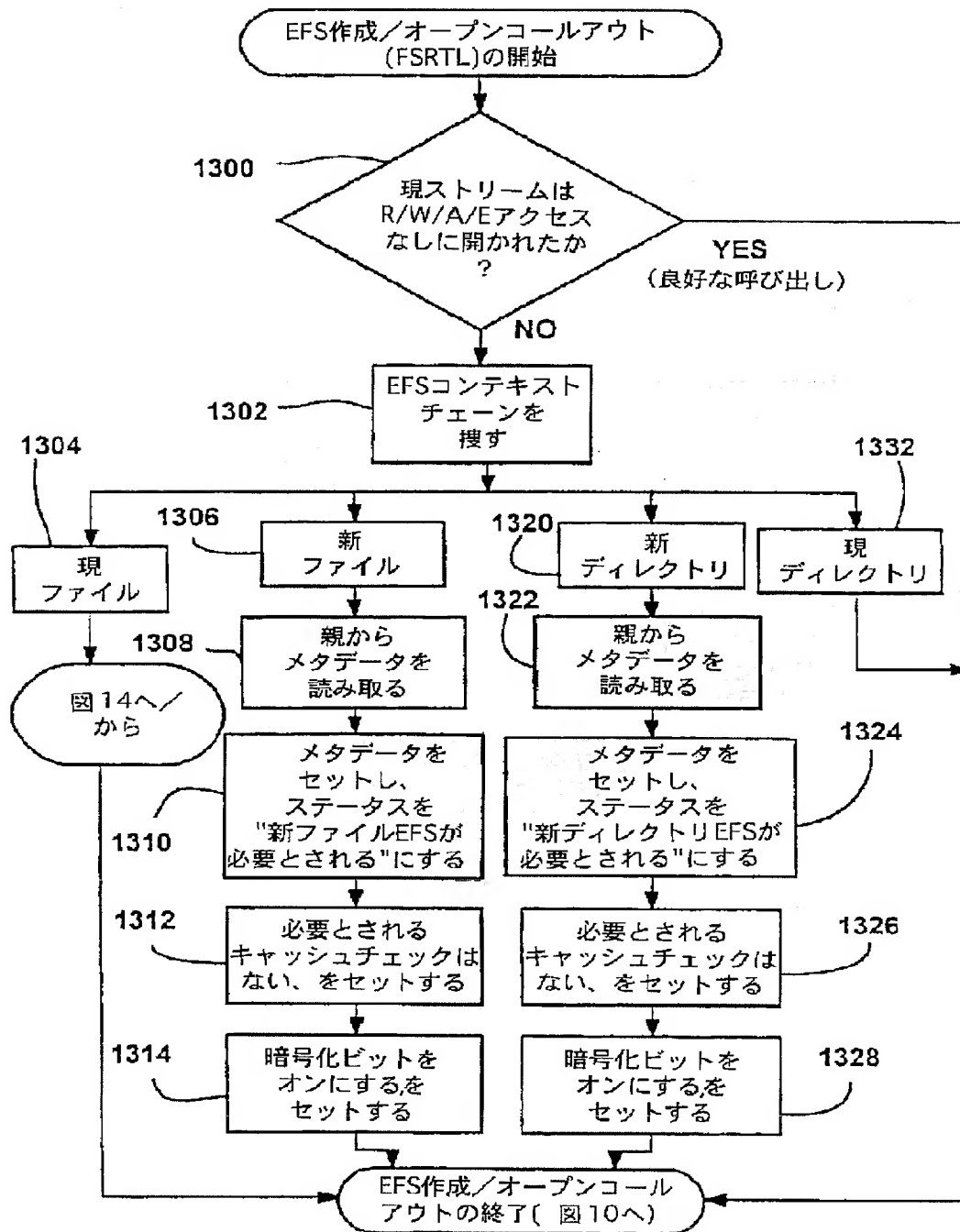
【 図 1 1 】



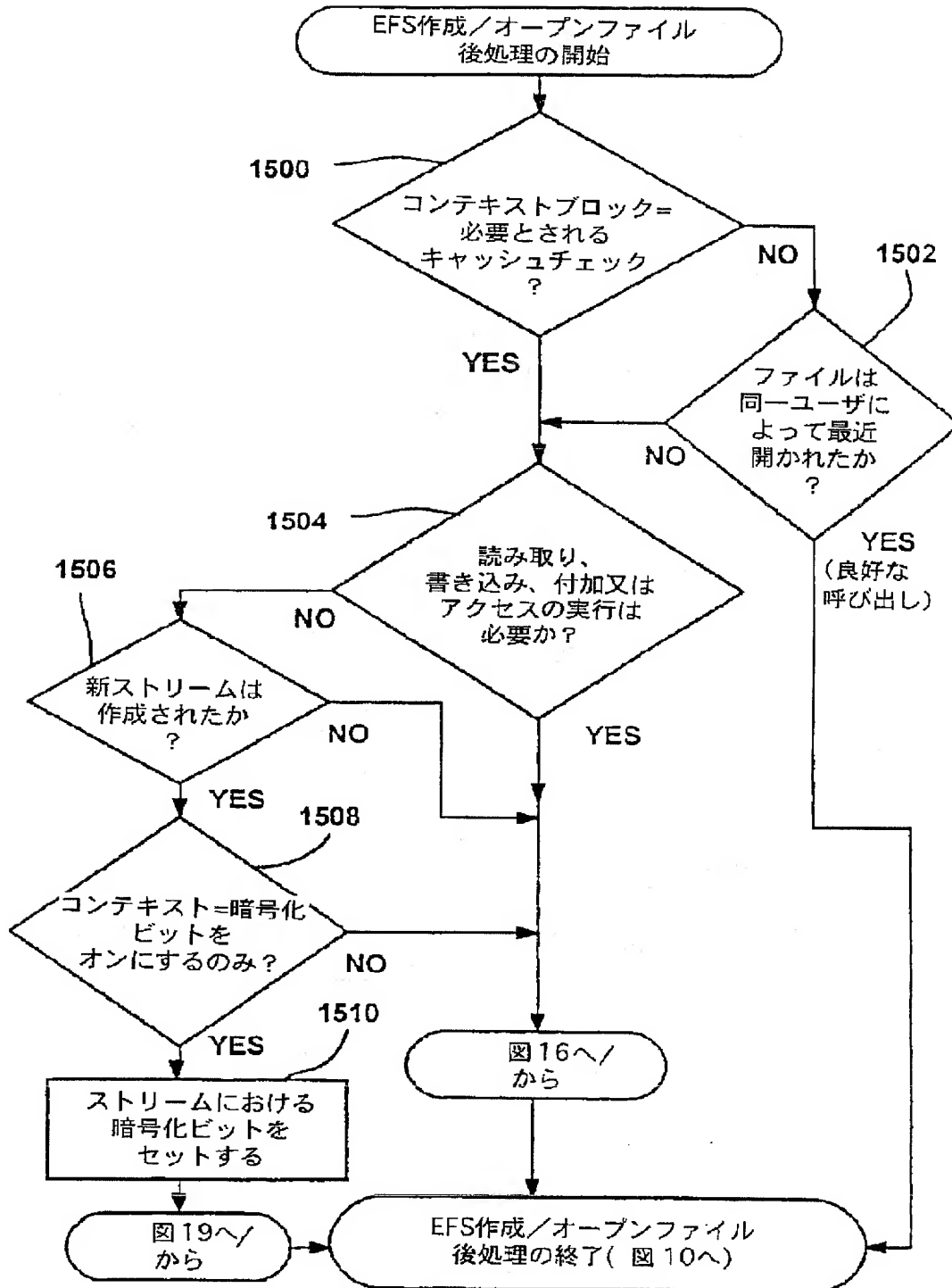
【 図 1 2 】



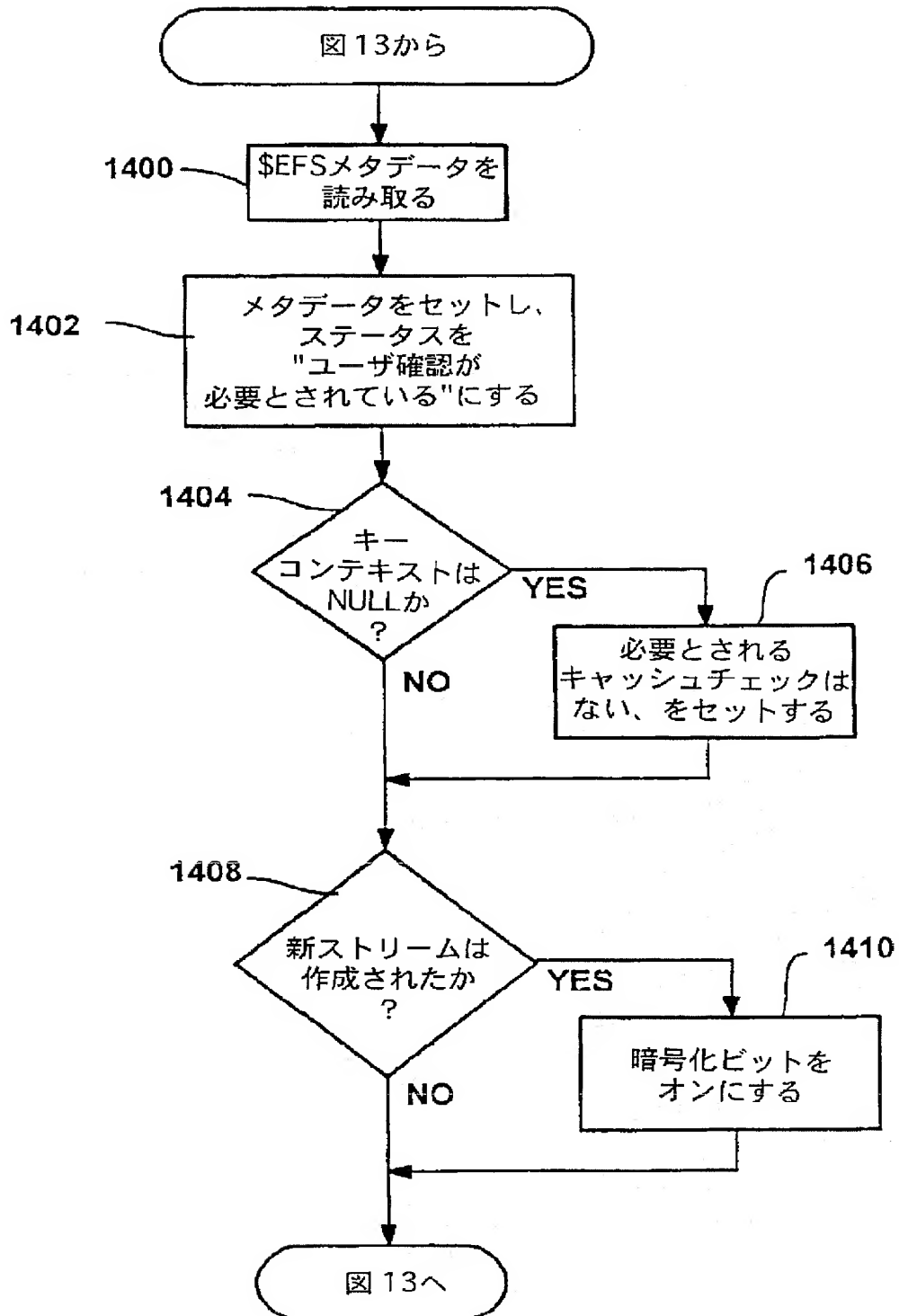
【 図 1 3 】



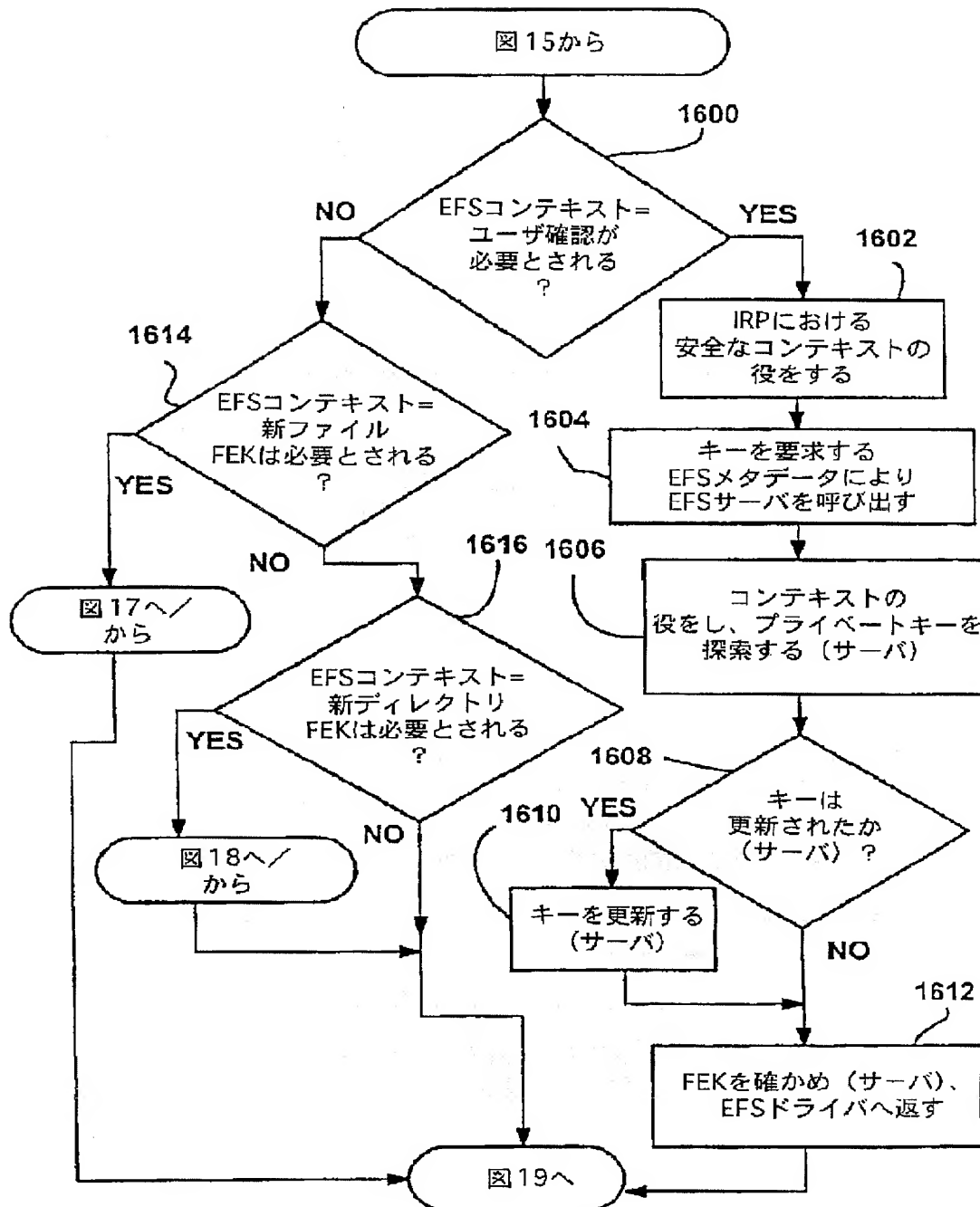
【 図 1 5 】



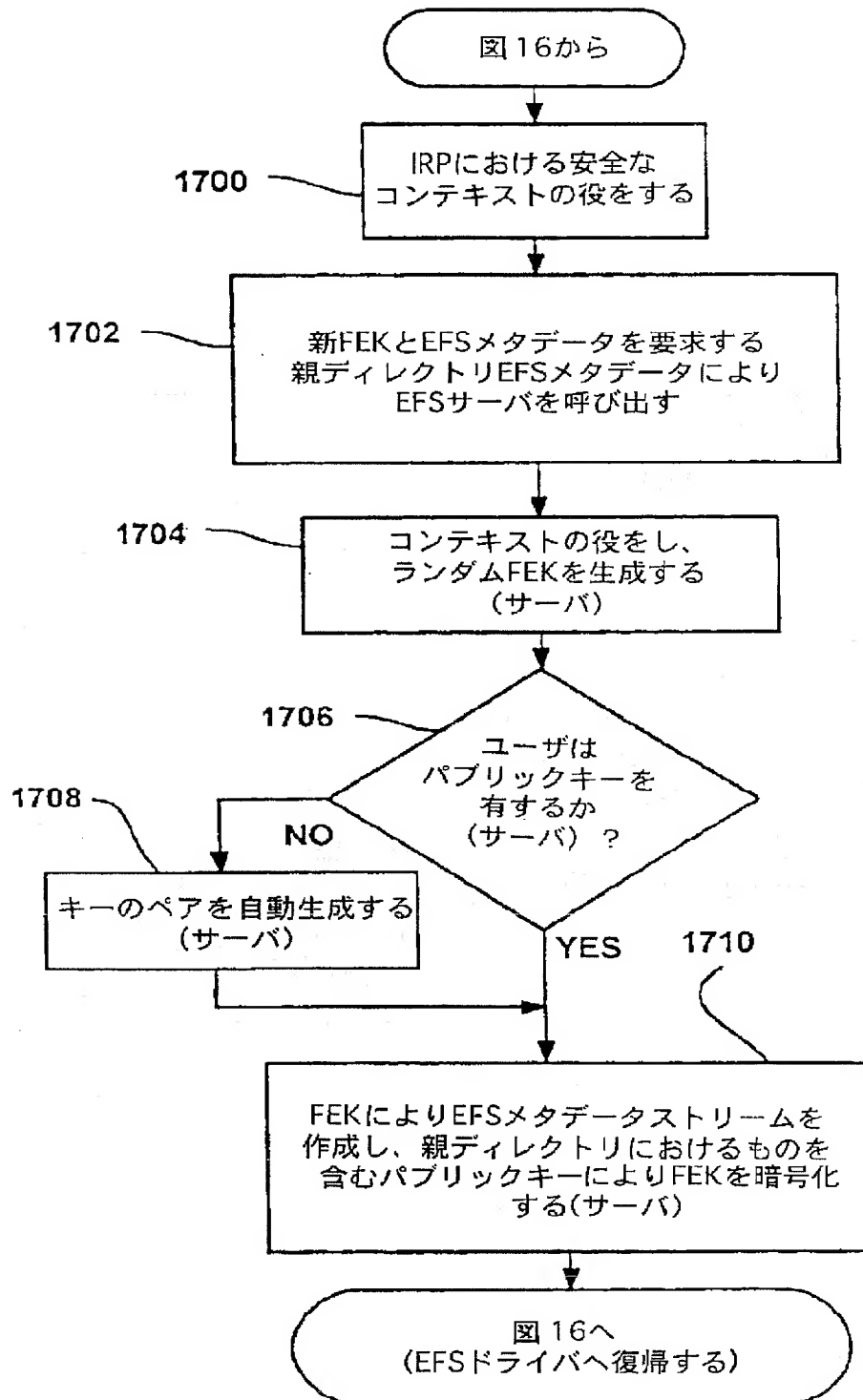
【 図 1 4 】



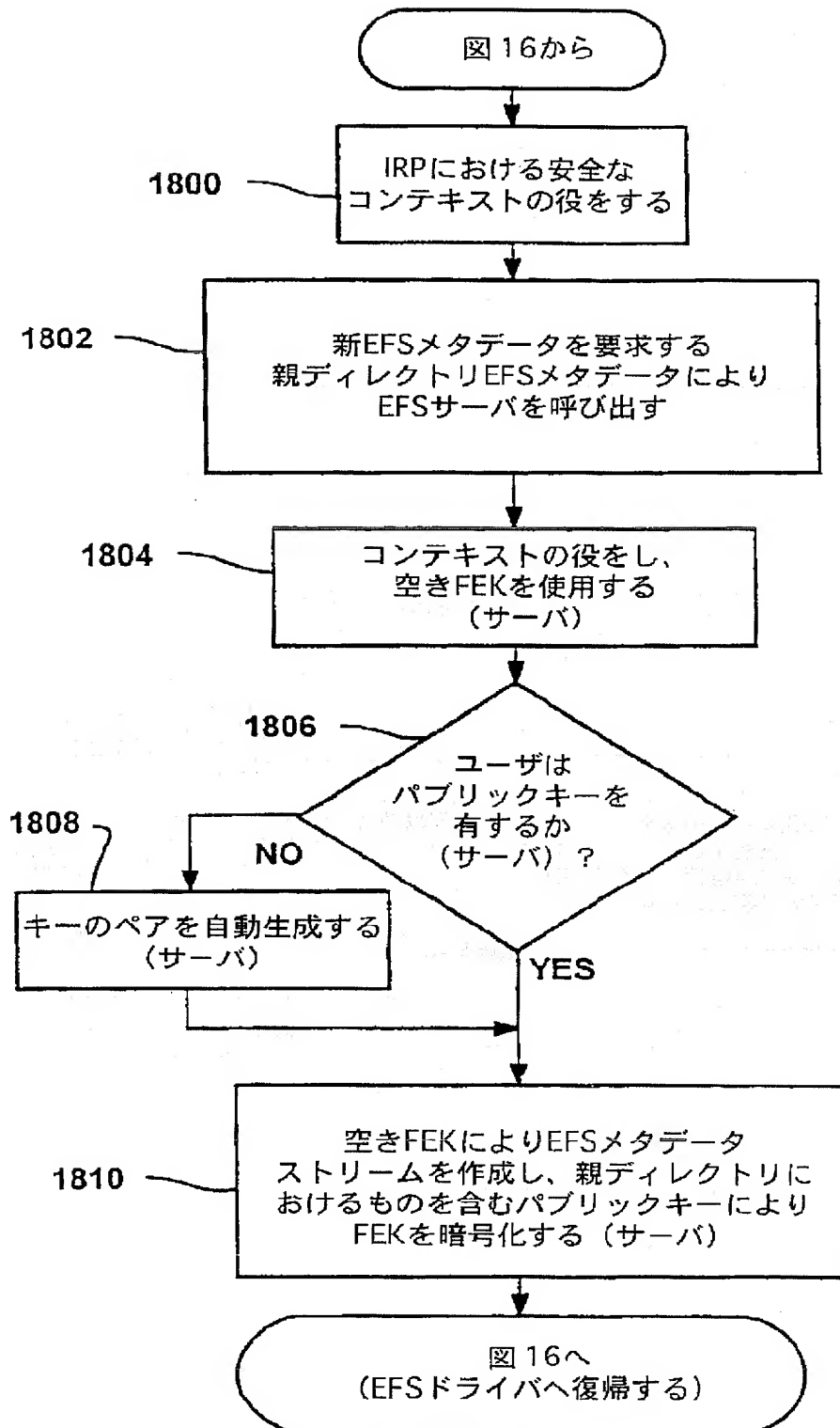
【 図 1 6 】



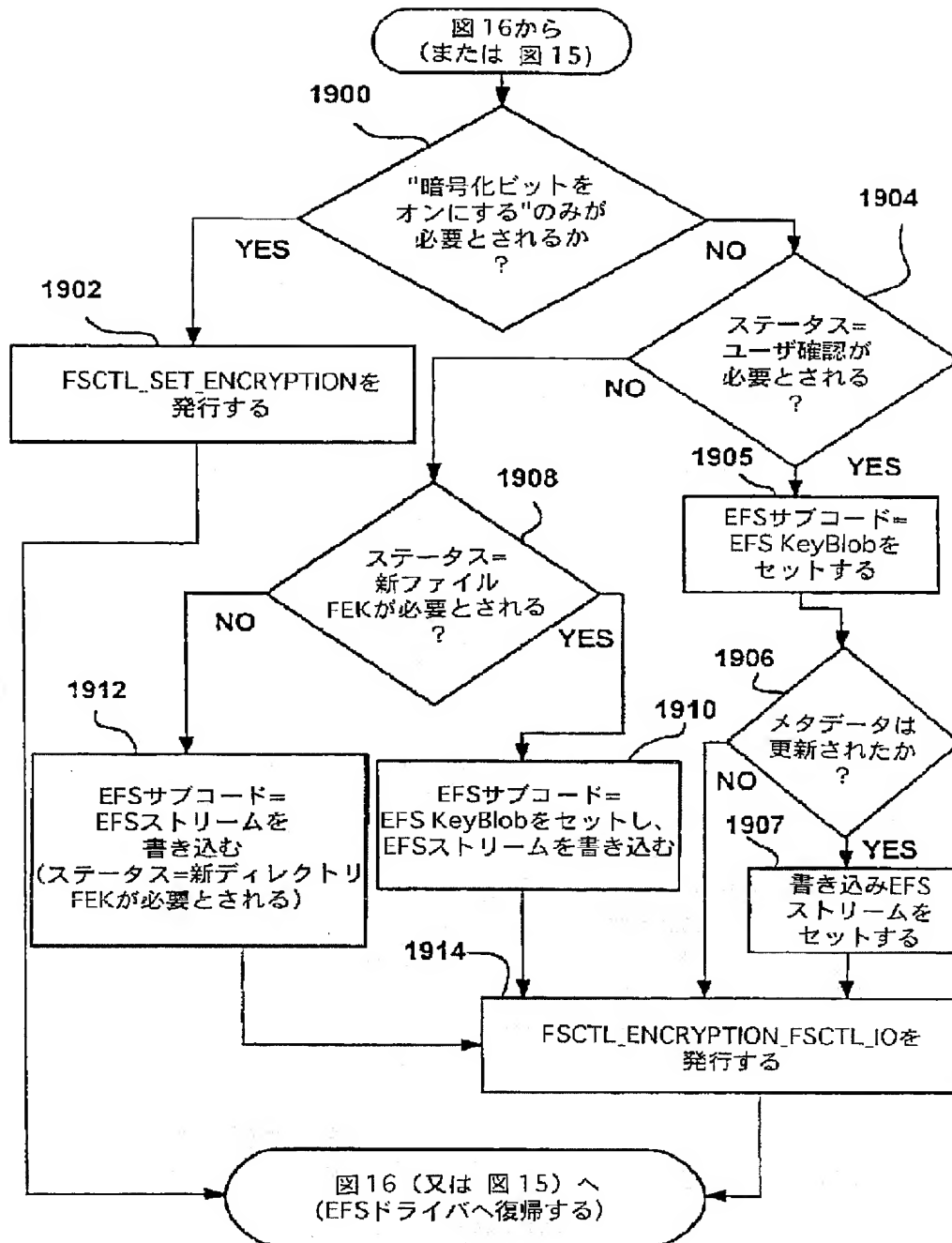
【 図 1 7 】



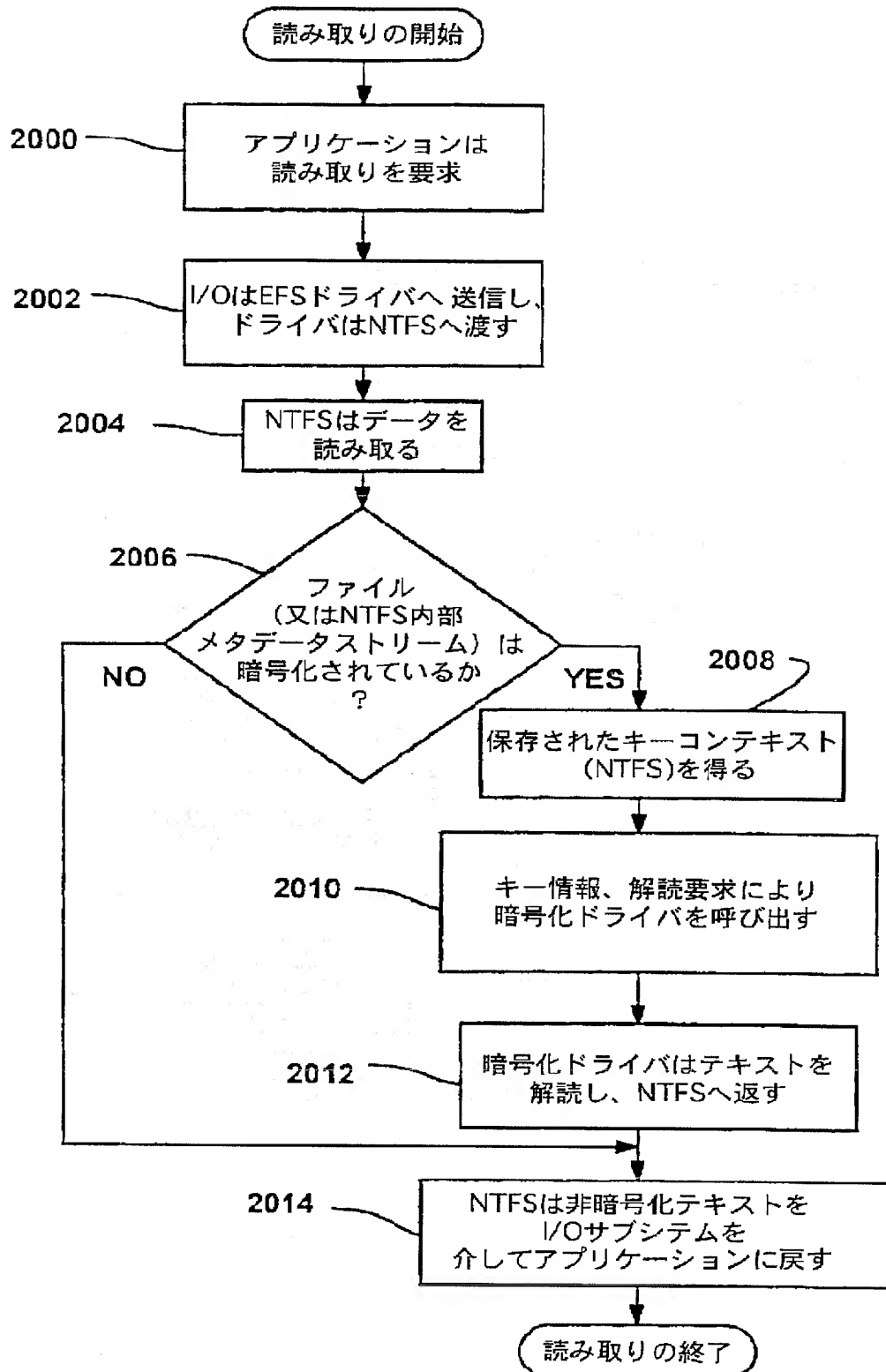
【 図 1 8 】



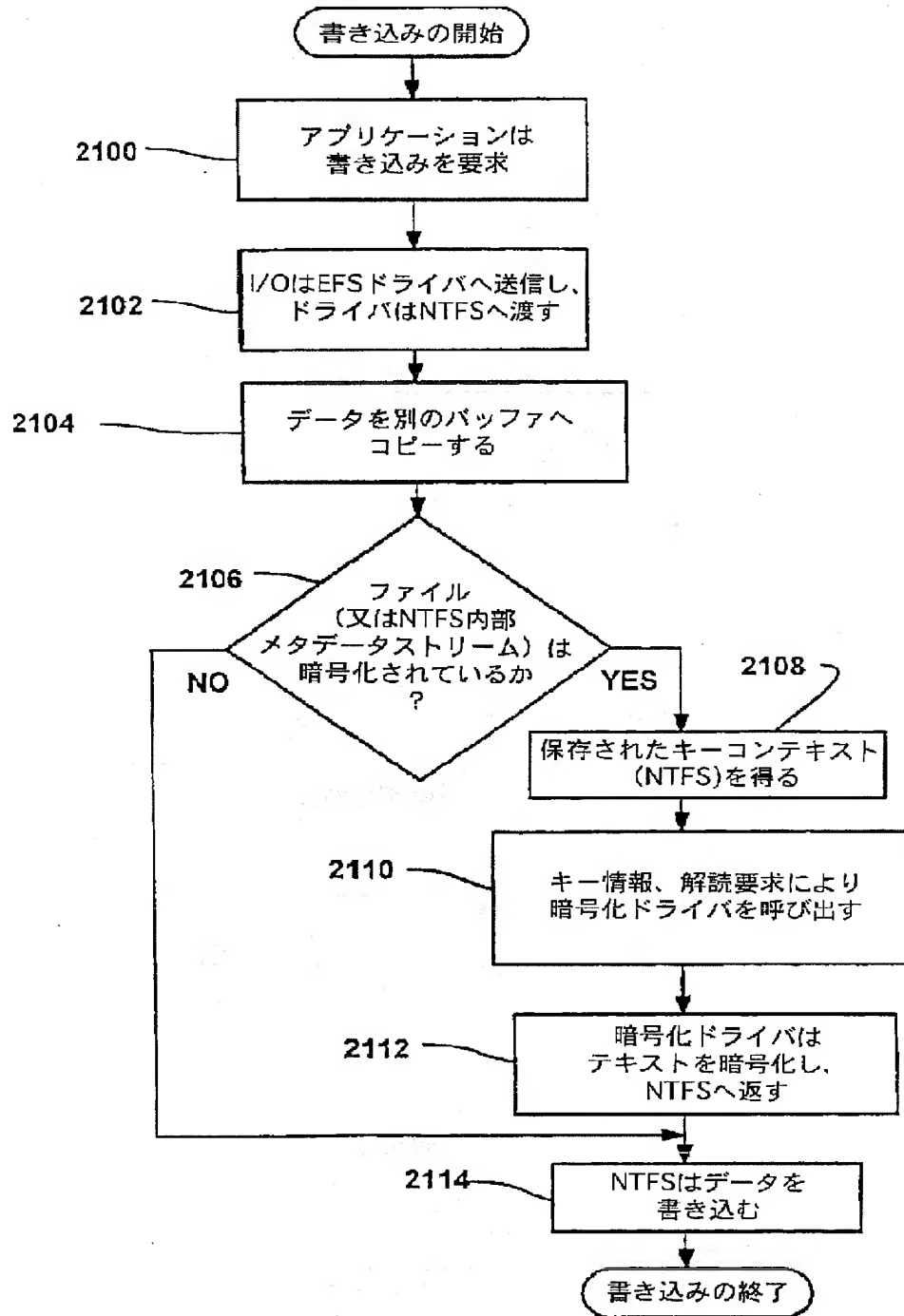
【図 19】



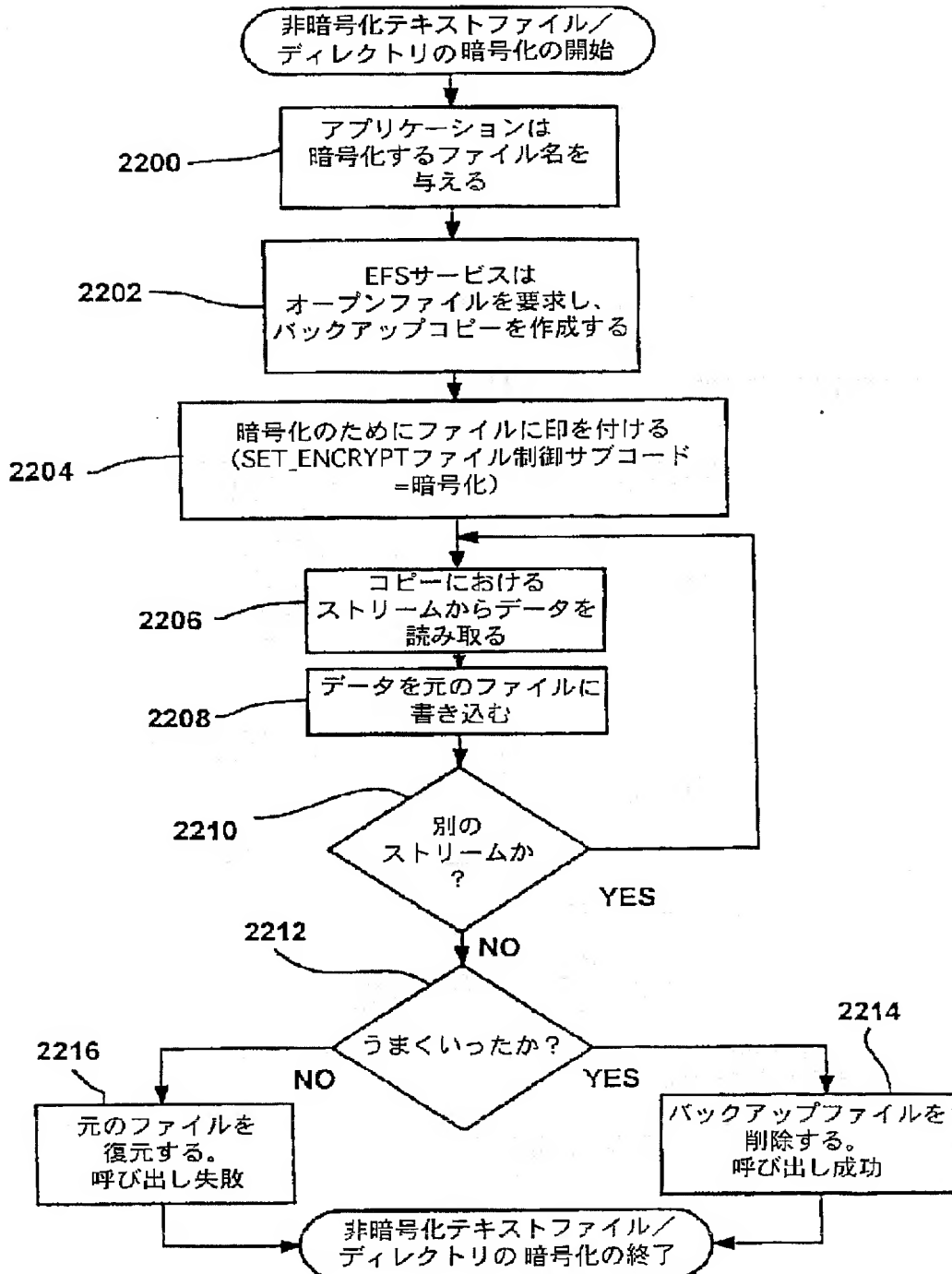
【 図 2 0 】



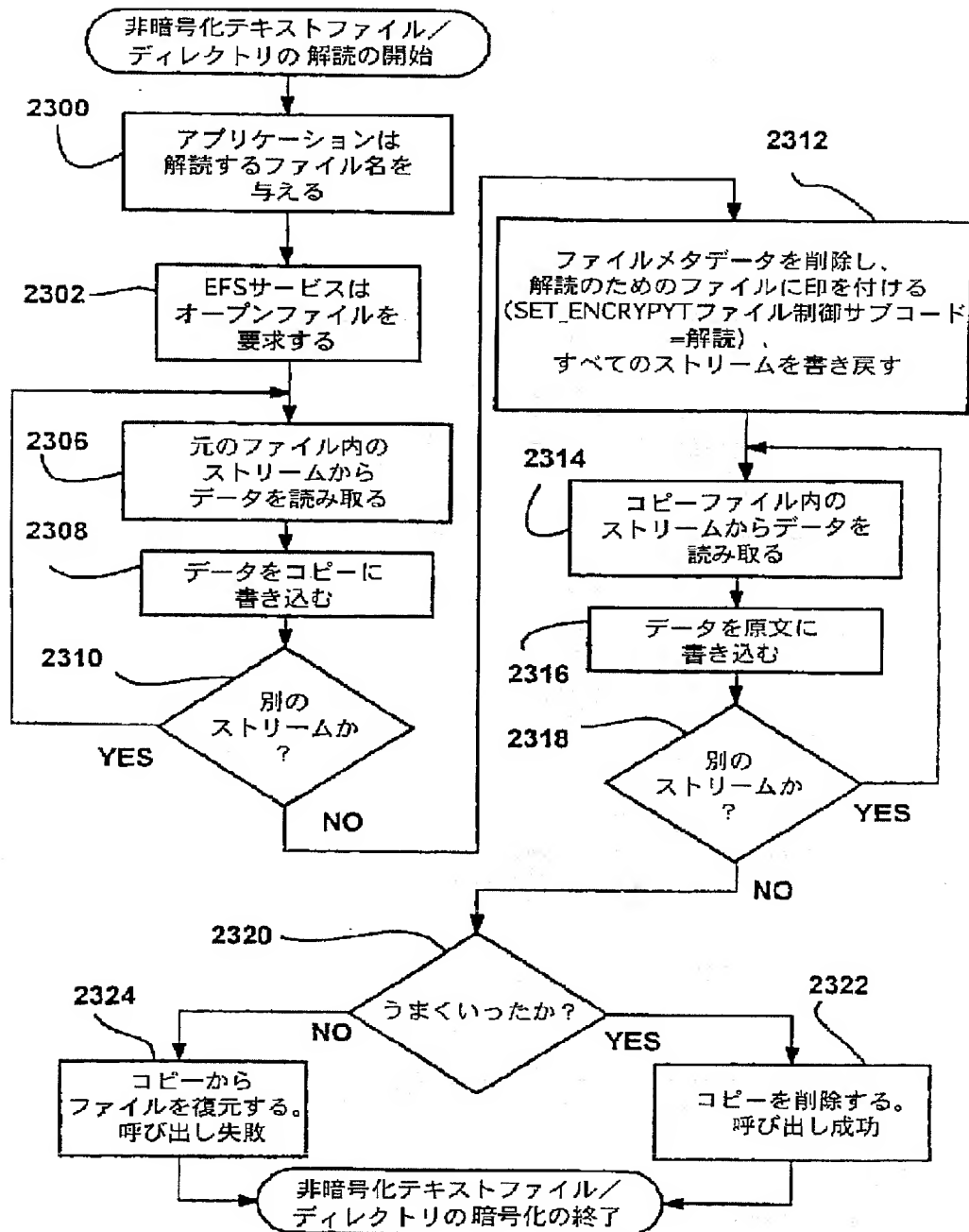
【 図 2 1 】



【 図 2 2 】



【 図 2 3 】



【 国際調査報告 】

INTERNATIONAL SEARCH REPORT

A. CLASSIFICATION OF SUBJECT MATTER IPC 6 G06F1/00		International Application No. PCT/US 98/19049
According to International Patent Classification (IPC) or to both national classification and IPC		
B. FIELDS SEARCHED Minimum documentation searched (classification system followed by classification symbols) IPC 6 G06F		
Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched		
Electronic data base consulted during the International search (name of data base and, where practical, search terms used)		
C. DOCUMENTS CONSIDERED TO BE RELEVANT		
Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	EP 0 681 233 A (IBM) 8 November 1995 see figures 1-5,7,14,15 see column 2, line 47 - column 6, line 57 see column 8, line 49 - column 13, line 48 --- -/-	1-5, 9-12,14, 17-20, 22,24, 25, 27-30,39
<input checked="" type="checkbox"/> Further documents are listed in the continuation of box C. <input checked="" type="checkbox"/> Patent family members are listed in annex.		
* Special categories of cited documents : "A" document defining the general state of the art which is not considered to be of particular relevance "E" earlier document but published on or after the international filing date "L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified) "O" document referring to an oral disclosure, use, exhibition or other means "P" document published prior to the international filing date but later than the priority date claimed "T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention "X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone "Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art. "&" document member of the same patent family		
Date of the actual completion of the international search 16 February 1999		Date of mailing of the international search report 23/02/1999
Name and mailing address of the ISA European Patent Office, P.B. 5518 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3018		Authorized officer Weiss, P

INTERNATIONAL SEARCH REPORT

International Application No.
PCT/US 98/19049

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT		
Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	<p>US 5 319 705 A (HALTER BERNARD J ET AL) 7 June 1994</p> <p>see figures 1-4,11,12 see figures 14-16 see column 7, line 44 - column 11, line 8 see column 23, line 51 - column 26, line 22</p>	<p>1-5, 9-11,14, 17-19, 22,24, 25,27,28</p>

INTERNATIONAL SEARCH REPORT

(information on patent family members)

International Application No.

PCT/US 98/19049

Patent document cited in search report		Publication date	Patent family member(s)	Publication date
EP 0681233	A	08-11-1995	US 5598470 A	28-01-1997
			CA 2145922 A	26-10-1995
			JP 7306780 A	21-11-1995
			US 5757908 A	26-05-1998
US 5319705	A	07-06-1994	JP 7093148 A	07-04-1995

フロントページの続き

- (72) 発明者 ガーグ プラエリット
アメリカ合衆国 98007 ワシントン州
ベルビュー ノース イースト 36 スト
リート ナンバー ジェイ-7 14557
- (72) 発明者 グ ジャンロング
アメリカ合衆国 98006 ワシントン州
ベルビュー サウス イースト 57 プレ
イス 16658
- (72) 発明者 ケリー ジェームズ ダブリュー. ジュ
ニア
アメリカ合衆国 98053 ワシントン州
レッドモンド ノース イースト 56 ス
トリート 22119
- (72) 発明者 カプラン キース エス.
アメリカ合衆国 98011 ワシントン州
ボーセル ノース イースト 95 プレイ
ス 20048
- (72) 発明者 レイチェル ロバート ピー.
アメリカ合衆国 98053 ワシントン州
レッドモンド ノース イースト 17 ス
トリート 20920
- (72) 発明者 アンドリュー ブライアン
アメリカ合衆国 98052 ワシントン州
レッドモンド ノース イースト 146
アベニュー 7706
- (72) 発明者 キムラ ゲイリー ディー.
アメリカ合衆国 98033 ワシントン州
カークランド ノース イースト 43 プ
レイス 11820
- (72) 発明者 ミラー トーマス ジェイ.
アメリカ合衆国 98004 ワシントン州
ベルビュー ノース イースト 47 スト
リート 9015

Fターム(参考) 5B017 AA03 BA07 CA16
5B076 FA00
5B082 GA11
5J104 AA16 EA04 EA06 EA19 JA21
NA03